

# The C-scape® Interface Management System

---

## Function Reference

LIANT SOFTWARE CORPORATION  
FRAMINGHAM, MASSACHUSETTS

The information in this manual is subject to change without notice. The information in this manual and the accompanying software are provided under the terms of a license agreement or non-disclosure agreement. The software may be used or copied only according to the terms of the agreement. No part of this manual may be reproduced, transmitted, transcribed, stored in any retrieval system, or translated into any language by any means without the prior written permission of:

Liant Software Corporation  
959 Concord Street  
Framingham, MA 01701  
U.S.A.

(508) 872-8700  
(508) 626-2221 *fax*

Copyright © 1986-92, by Liant Software Corporation.  
All rights reserved.

## **Trademarks**

C-scape and Look & Feel are registered trademarks of Liant Software Corporation. Other trademarks are property of their respective companies.



# Contents

---

<b>Acknowledgements</b>	ii
<b>Contents</b>	iii
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Conventions .....	1
<b>Chapter 2 C-scape routines</b>	<b>3</b>
2.1 Menu functions .....	3
2.2 Sed activation functions .....	4
2.3 Sed paint functions .....	4
2.4 Sed parameter functions .....	4
2.5 Sed border functions .....	6
2.6 Sed cursor movement functions .....	6
2.7 Sed field movement functions .....	7
2.8 Sed field functions .....	7
2.9 Popup functions .....	10
2.10 Menuing system functions .....	10
2.11 String formatting functions .....	10
2.12 Date and time functions .....	11
2.13 Bob functions .....	12
2.14 Sled functions .....	12

2.15	Ted parameter functions .....	12
2.16	Ted editing functions .....	13
2.17	Ted block functions .....	14
2.18	Ted I/O functions .....	14
2.19	Screen file functions .....	15
2.20	Help functions .....	15
2.21	Keyboard functions .....	15
2.22	Memory allocation functions .....	16

## **Appendix A    Function Reference**

## **Appendix B    Field Function Reference**

## **Appendix C    Error Messages**

# Chapter 1 Introduction

C-scape's documentation comes in two volumes. This is the second volume, the *C-scape Function Reference*. It contains a reference page for each C-scape routine and each standard field function. It also contains a listing of the C-scape error messages.

The first volume, the *C-scape Manual*, provides a conceptual overview of C-scape. Refer to it for a theoretical and practical discussion of C-scape's features.

These are the chapters and appendices in this reference:

Chapter 2      *C-scape Routines*

This chapter lists the C-scape library routines grouped by category.

Appendix A    *Function Reference*

This appendix contains reference pages for every C-scape routine. These pages are organized alphabetically. For each routine, the reference page provides a listing of its arguments, return values, a brief description of its uses, an example of its use in a C-scape application, and any other information or warnings.

Appendix B    *Field Function Reference*

This appendix contains a complete reference page for each of the standard field functions. Each reference page provides a description of the field function's characteristics, variable type, internal structure, source file, and an example of its use.

Appendix C    *Error Messages*

This appendix contains a list of the C-scape error messages and a description of each one.

## 1.1 Conventions

Throughout this manual we use the following naming conventions:


The term "display" describes the actual hardware display device (terminal, monitor, CRT).

The term “screen” describes a collection of characters grouped together on the display for some distinct purpose (a “help” screen, a “data entry” screen).

The word “menu” indicates a C-scape data object (a menu object) as well as a means for presenting user choices (a “123 menu”).

The meaning of these terms should be clear from their context.

The following typeface conventions are used:

<b>boldface</b>	Indicates function names.
<i>italics</i>	Indicates variables, function arguments, and file names.
fixed print	Is used for code examples.
	Indicates a key on the keyboard.

The following data types are used throughout this reference and the C-scape source code:

VOID *	Is #defined to <b>void *</b> for ANSI C compilers and to <b>char *</b> for older compilers. This data type is used wherever a general purpose data pointer is needed. C-scape uses this whenever a <b>void *</b> would normally be used.
SIZE_T	Is #defined to <b>size_t</b> for ANSI C compilers and to <b>unsigned int</b> for older compilers.

Within source code examples, /\* ... \*/ indicates a section of code not listed.

## Chapter 2 **C-scape routines**

C-scape comes with over 300 library routines. These are functions and macros that you can call from within your programs to perform a wide variety of tasks—including windowing, menuing, text editing, data entry, mouse manipulation, and lower level functions.

C-scape actually consists of two libraries: the C-scape Library and the Oakland Windowing Library (OWL). This volume is a reference for the functions in the C-scape library. For information on the functions in OWL, consult the *OWL Function Reference*.

This chapter contains a listing of C-scape routines by purpose. Each section lists functions that perform similar or related tasks. The order of sections in this chapter generally parallels the order in which concepts are discussed in *The C-scape Manual*.

### **2.1 Menu functions**

<b>menu_Flush</b>	Frees the menu object's creation data.
<b>menu_GetCol</b>	Gets the current menu column.
<b>menu_GetFieldCol</b>	Gets the leftmost column of a field.
<b>menu_GetFieldRow</b>	Gets the row of a field.
<b>menu_GetHeight</b>	Gets the current menu's height.
<b>menu_GetRow</b>	Gets the current menu's row.
<b>menu_GetWidth</b>	Gets the current menu's width.
<b>menu_Open</b>	Creates a new menu object.
<b>menu_Printf</b>	Defines a menu object.
<b>menu_SetWrapWidth</b>	Sets the menu's word wrap width.
<b>menu_UnPrintf</b>	Gets a format string for a field.

## 2.2 Sed activation functions

<b>sed_Alloc</b>	Allocates variable storage for a sed.
<b>sed_Close</b>	Destroys a sed object.
<b>sed_CreateBob</b>	Creates a bob object from a sed.
<b>sed_Go</b>	Activates a sed.
<b>sed_Ok</b>	Checks if the sed is valid.
<b>sed_Open</b>	Creates a new sed object.
<b>sed_SetMouse</b>	Attaches a mouse handler to a sed.
<b>sed_SetMouseCode</b>	Sets the sed's mousecode.
<b>sed_SetNextWin</b>	Passes control to another window.

## 2.3 Sed paint functions

<b>sed_Pop</b>	Fires a sed's window.
<b>sed_Repaint</b>	Repaints the sed.
<b>sed_RepaintField</b>	Repaints a field.
<b>sed_RepaintFields</b>	Repaints all the fields.
<b>sed_Top</b>	Moves the sed's window to the foreground.
<b>sed_Update</b>	Paints a sed without converting data.
<b>sed_UpdateCurrField</b>	Paints the current field.
<b>sed_UpdateField</b>	Paints a field without converting data.
<b>sed_UpdateFields</b>	Paints all the fields.

## 2.4 Sed parameter functions

<b>sed_Center</b>	Centers a sed.
<b>sed_GetBaton</b>	Gets the sed's baton.
<b>sed_GetColors</b>	Gets the sed's colors.
<b>sed_GetCorners</b>	Gets the sed's corners.
<b>sed_GetCursorType</b>	Gets the sed's current cursor type.

<b>sed_GetData</b>	Gets the sed's generic data pointer.
<b>sed_GetExit</b>	Gets the sed's exit state.
<b>sed_GetHeight</b>	Gets the sed's height.
<b>sed_GetLabel</b>	Gets the value of the sed's label.
<b>sed_GetMenu</b>	Gets the sed's menu.
<b>sed_GetMenuHeight</b>	Gets the height of the sed's menu.
<b>sed_GetMenuWidth</b>	Gets the width of the sed's menu.
<b>sed_GetPosition</b>	Gets the sed's position.
<b>sed_GetScratchPad</b>	Gets the sed's scratch pad.
<b>sed_GetScratchSize</b>	Gets the size of the sed's scratch pad.
<b>sed_GetSize</b>	Gets the sed's dimensions.
<b>sed_GetWidth</b>	Gets the sed's width.
<b>sed_GetXoffset</b>	Gets the sed's xoffset.
<b>sed_GetYoffset</b>	Gets the sed's yoffset.
<b>sed_IsActive</b>	Checks if the sed is in active mode.
<b>sed_SetActive</b>	Sets the sed's active flag.
<b>sed_SetAux</b>	Attaches an auxiliary function to a sed.
<b>sed_SetBaton</b>	Sets the value of the sed's baton.
<b>sed_SetColors</b>	Sets the sed's colors.
<b>sed_SetCursorType</b>	Sets the sed's cursor type.
<b>sed_SetData</b>	Sets the sed's generic data pointer.
<b>sed_SetExit</b>	Sets the sed's exit state.
<b>sed_SetExplode</b>	Attaches an explode function to the sed.
<b>sed_SetHeight</b>	Sets the sed's height.
<b>sed_SetLabel</b>	Sets the value of the sed's label.
<b>sed_SetPosition</b>	Sets the sed's position.
<b>sed_SetShadow</b>	Sets the sed shadow size.
<b>sed_SetShadowAttr</b>	Sets the shadow attribute of a sed.
<b>sed_SetSpecial</b>	Attaches the sed's special function.
<b>sed_SetWidth</b>	Sets the sed's width.
<b>sed_ToggleExit</b>	Exits the sed.

## 2.5 Sed border functions

<b>sed_BorderExists</b>	Checks if the sed has a border.
<b>sed_BorderPrompt</b>	Displays a prompt in the border.
<b>sed_GetBordCorners</b>	Gets the sed's border's corners.
<b>sed_GetBorderColor</b>	Gets the color of the sed's border.
<b>sed_GetBorderHeight</b>	Gets the height of the sed's border.
<b>sed_GetBorderWidth</b>	Gets the width of the sed's border.
<b>sed_RedirectPrompt</b>	Redirects a border prompt.
<b>sed_RepaintBorder</b>	Repaints the sed's border.
<b>sed_SetBorder</b>	Attaches a border to a sed.
<b>sed_SetBorderColor</b>	Sets the color of the sed's border.
<b>sed_SetBorderFeature</b>	Sets the features of a border.
<b>sed_SetBorderTitle</b>	Sets the title of a titled border.

## 2.6 Sed cursor movement functions

<b>sed_DecChar</b>	Moves to the previous character.
<b>sed_GoEnd</b>	Goes to the end of the current field.
<b>sed_GoHome</b>	Goes to the start of the current field.
<b>sed_GotoChar</b>	Goes to a position in the current field.
<b>sed_IncChar</b>	Moves to the next character in a field.
<b>sed_IsEnd</b>	Checks if the cursor is at the end of a field.
<b>sed_IsHome</b>	Checks if the cursor is at the start of a field.
<b>sed_Overwrite</b>	Overwrites the current character.
<b>sed_PullLeft</b>	Deletes a character, pulling left.
<b>sed_PullRight</b>	Deletes a character, pulling right.
<b>sed_PushLeft</b>	Inserts a character, pushing left.
<b>sed_PushRight</b>	Inserts a character, pushing right.



## 2.7 Sed field movement functions

<b>sed_DecField</b>	Goes to the previous field.
<b>sed_DownField</b>	Moves down a field.
<b>sed_GotoField</b>	Goes to a specific field.
<b>sed_GotoFirstField</b>	Goes to the first field.
<b>sed_GotoGridField</b>	Goes to a field using its grid address.
<b>sed_GotoLastField</b>	Goes to the last field.
<b>sed_GotoNameField</b>	Goes to a field using its name.
<b>sed_IncField</b>	Goes to the next field.
<b>sed_LeftField</b>	Moves left a field.
<b>sed_MoveField</b>	Moves a field.
<b>sed_PageDown</b>	Scrolls the sed down one page.
<b>sed_PageLeft</b>	Scrolls the sed left one page.
<b>sed_PageRight</b>	Scrolls the sed right one page.
<b>sed_PageUp</b>	Scrolls the sed up one page.
<b>sed_RightField</b>	Moves right a field.
<b>sed_ScrollDown</b>	Scrolls the sed down.
<b>sed_ScrollLeft</b>	Scrolls the sed left.
<b>sed_ScrollRight</b>	Scrolls the sed right.
<b>sed_ScrollUp</b>	Scrolls the sed up.
<b>sed_UpField</b>	Moves up a field.

## 2.8 Sed field functions

<b>inter_field</b>	Controls field function inter-field movement.
<b>inter_field_grid</b>	Controls field function inter-field grid movement.
<b>inter_page</b>	Controls field function inter-page movement.
<b>sed_DeleteField</b>	Removes a field.
<b>sed_DeleteRows</b>	Deletes rows from a sed.

<b>sed_DoAux</b>	Calls a sed's auxiliary function.
<b>sed_DoFieldFenter</b>	Executes the field's <b>fenter</b> function.
<b>sed_DoFieldFexit</b>	Executes the field's <b>fexit</b> function.
<b>sed_DoFieldFkey</b>	Executes the field's <b>fkey</b> function.
<b>sed_DoFieldSenter</b>	Executes the field's <b>senter</b> function.
<b>sed_DoFieldSexit</b>	Executes the field's <b>sexit</b> function.
<b>sed_DoSenter</b>	Executes all the fields' <b>senter</b> functions.
<b>sed_DoSexit</b>	Executes all the fields' <b>sexit</b> functions.
<b>sed_DoSpecial</b>	Executes the sed's special function.
<b>sed_GetChar</b>	Gets a character within the current field.
<b>sed_GetCurrChar</b>	Gets the character at the current position.
<b>sed_GetCurrMerge</b>	Gets the current field's merge.
<b>sed_GetCurrRecord</b>	Gets the current field's record.
<b>sed_GetCurrRecordLen</b>	Gets current field's record length.
<b>sed_GetCurrVar</b>	Gets the current field's variable.
<b>sed_GetFieldBob</b>	Gets the bob attached to a field.
<b>sed_GetFieldChar</b>	Gets a character within a field.
<b>sed_GetFieldCol</b>	Gets the field's first column number.
<b>sed_GetFieldColors</b>	Gets the field's colors.
<b>sed_GetFieldCount</b>	Gets the number of fields in the sed.
<b>sed_GetFieldData</b>	Gets a field's data pointer.
<b>sed_GetFieldDataCount</b>	Gets the field's number of data pointers.
<b>sed_GetFieldLastCol</b>	Gets the field's last column number.
<b>sed_GetFieldName</b>	Gets a field's name.
<b>sed_GetFieldNo</b>	Gets the current field number.
<b>sed_GetFieldRow</b>	Gets the field's row number.
<b>sed_GetFieldWidth</b>	Gets the field's displayed width.
<b>sed_GetFieldXoffset</b>	Gets the field's xoffset.
<b>sed_GetFuncs</b>	Gets the field's field function.
<b>sed_GetGridCol</b>	Gets the field's grid column number.
<b>sed_GetGridField</b>	Gets the field number at the grid position.

<b>sed_GetGridRow</b>	Gets the field's grid row number.
<b>sed_GetMerge</b>	Gets the field's merge.
<b>sed_GetMergeLen</b>	Gets the length of the field's merge.
<b>sed_GetMergePos</b>	Gets the current position in the merge.
<b>sed_GetNameNo</b>	Gets the field number associated with a name.
<b>sed_GetNameVar</b>	Gets the field variable associated with a name.
<b>sed_GetRecord</b>	Gets the field's record.
<b>sed_GetRecordLen</b>	Gets the length of the field's record.
<b>sed_GetRecordPos</b>	Gets the current position in the record.
<b>sed_GetVar</b>	Gets the field's variable.
<b>sed_GetVarSize</b>	Gets the field's variable size.
<b>sed_InsertRows</b>	Inserts rows in a sed.
<b>sed_IsMarkedField</b>	Checks if the field is marked.
<b>sed_IsProtectedField</b>	Checks if the field is protected.
<b>sed_MarkField</b>	Marks a field.
<b>sed_ProtectField</b>	Protects a field.
<b>sed_SearchMerge</b>	Searches for fields by first letter.
<b>sed_SetCurrRecord</b>	Sets the current field's record.
<b>sed_SetFieldBob</b>	Attaches a bob object to a field.
<b>sed_SetFieldData</b>	Sets a field's data pointer.
<b>sed_SetFieldName</b>	Sets a field's name.
<b>sed_SetFieldWidth</b>	Sets the field's displayed width.
<b>sed_SetFuncs</b>	Sets the field's field function.
<b>sed_SetRecord</b>	Sets the field's record.
<b>sed_SetVar</b>	Sets the field's variable.
<b>sed_SwapFields</b>	Swaps fields' positions.
<b>sed_UnMarkField</b>	Unmarks a field.
<b>sed_UnProtectField</b>	Unprotects a field.

## 2.9 Popup functions

<b>opc_Close</b>	Destroys an opc structure.
<b>opc_Edit</b>	Creates a popup editor box.
<b>opc_FileBox</b>	Displays a file selection box.
<b>opc_Menu</b>	Creates a popup menu.
<b>opc_Message</b>	Creates a popup message box.
<b>opc_Open</b>	Creates an opc structure.
<b>opc_Prompt</b>	Creates a popup prompt box.
<b>opc_Text</b>	Creates a popup text box.
<b>opc_Verify</b>	Displays a verification box.
<b>opc_View</b>	Views text in a popup box.

## 2.10 Menuing system functions

<b>frame_Close</b>	Destroys a frame menu object.
<b>frame_Go</b>	Activates a frame menu object.
<b>frame_Lock</b>	Locks a frame menu choice.
<b>frame_Open</b>	Creates a frame menu object.
<b>frame_Repaint</b>	Paints a frame menu object.
<b>frame_SetPosition</b>	Sets the position of a frame menu.
<b>frame_UnLock</b>	Unlocks a frame menu choice.
<b>slug_Close</b>	Destroys a slug menu object.
<b>slug_Go</b>	Activates a slug menu object.
<b>slug_Open</b>	Creates a slug menu object.
<b>slug_Repaint</b>	Paints a slug menu object.

## 2.11 String formatting functions

<b>strcenter</b>	Centers a string.
<b>strclip</b>	Removes the trailing spaces from a string.

<b>strfill</b>	Fills a string with characters.
<b>strleft</b>	Left justifies a string.
<b>strpad</b>	Pads a string with spaces.
<b>strpreclip</b>	Removes the leading spaces from a string.
<b>strright</b>	Right justifies a string.
<b>strwrap</b>	Word-wraps a string.

## 2.12 Date and time functions

<b>ott_Init</b>	Initialize a time.
<b>ott_Now</b>	Get the current time.
<b>tm_AddDays</b>	Adds a number of days to a time.
<b>tm_AddSecs</b>	Adds a number of seconds to a time.
<b>tm_Adjust</b>	Adjusts all the elements of a time structure until they are valid.
<b>tm_Cmp</b>	Compares two times.
<b>tm_Copy</b>	Copies one time to another.
<b>tm_DayOfWeek</b>	Returns the week day of a given date.
<b>tm_DaysInMonth</b>	Returns the number of days in a given month.
<b>tm_ElapDays</b>	Subtracts two times and give the result in days.
<b>tm_ElapSecs</b>	Subtracts two times and give the result in seconds.
<b>tm_Init</b>	Initialize a time.
<b>tm_IsDateValid</b>	Checks the validity of the date portion of a time structure.
<b>tm_IsTimeValid</b>	Checks the validity of the time portion of a time structure.
<b>tm_IsValid</b>	Checks the validity of a time structure.
<b>tm_Now</b>	Places the current time into a time structure.
<b>tm_StructToTt</b>	Convert a tm struct to a TIME_T value.
<b>tm_TtToStruct</b>	Convert a TIME_T value to a tm struct.
<b>tm_Zero</b>	Sets all the elements of a time structure to zero.

## 2.13 Bob functions

<b>bob_GetOwner</b>	Gets the sed that owns a bob.
<b>bob_GetPosition</b>	Gets a bob object's position.
<b>bob_GetSed</b>	Gets the bob object's sed.
<b>bob_GetSize</b>	Gets a bob object's size.
<b>bob_Go</b>	Activates a bob object.
<b>bob_Pop</b>	Removes a bob object from the display.
<b>bob_Repaint</b>	Paints a bob object.
<b>bob_SetPosition</b>	Sets a bob object's position.
<b>sed_GetAncestor</b>	Gets a sed's most distant ancestor.
<b>sed_GetBob</b>	Gets the bob created from the sed.

## 2.14 Sled functions

<b>sled_DeleteRows</b>	Destroys rows in a sled.
<b>sled_GetColSize</b>	Gets the size of the sled's columns.
<b>sled_GetColVar</b>	Gets a pointer to a sled entry.
<b>sled_GetRow</b>	Gets the sled's current row.
<b>sled_InsertRows</b>	Inserts blank rows into a sled.
<b>sled_IsLastRow</b>	Checks if the current row is the last.
<b>sled_Open</b>	Creates a scrolling list.
<b>sled_Remap</b>	Remaps the sled's field variables.
<b>sled_Repaint</b>	Refreshes the sled to the display.
<b>sled_SetColVar</b>	Sets the value of a sled entry.

## 2.15 Ted parameter functions

<b>ted_GetCursor</b>	Gets the text cursor offset.
<b>ted_GetInsert</b>	Gets the text insert mode.
<b>ted_GetLineLen</b>	Gets the length of the current line.

<b>ted_GetMark</b>	Gets the text marking mode.
<b>ted_GetMaxSize</b>	Gets the text buffer's maximum size.
<b>ted_GetNewLineChar</b>	Gets the displayed newline character.
<b>ted_GetPosition</b>	Gets the text position in the sed.
<b>ted_GetRefresh</b>	Gets the text refresh mode.
<b>ted_GetSize</b>	Gets the text buffer's size.
<b>ted_GetString</b>	Copies a string from the sed.
<b>ted_GetTabChar</b>	Gets the displayed tab character.
<b>ted_GetTabSize</b>	Gets the sed's tab size.
<b>ted_GetWrapWidth</b>	Gets the sed's wrap width.
<b>ted_SetInsert</b>	Sets the text insert mode.
<b>ted_SetMark</b>	Sets the text marking mode.
<b>ted_SetMaxSize</b>	Sets the text buffer's maximum size.
<b>ted_SetMoveMethod</b>	Sets the cursor movement method.
<b>ted_SetNewLineChar</b>	Sets the displayed newline character.
<b>ted_SetRefresh</b>	Sets the text refresh mode.
<b>ted_SetTabChar</b>	Sets the displayed tab character.
<b>ted_SetTabSize</b>	Sets the sed's tab size.
<b>ted_SetWrapWidth</b>	Sets the sed's wrap width.

## 2.16 Ted editing functions

<b>ted_AddChar</b>	Writes a character to the sed.
<b>ted_AddRow</b>	Adds a row to the sed.
<b>ted_AddString</b>	Writes a string to the sed.
<b>ted_DeleteChar</b>	Deletes a character.
<b>ted_DownChar</b>	Moves down one character.
<b>ted_GoBottom</b>	Moves to the bottom of the sed text.
<b>ted_GoEnd</b>	Moves to the end of the row.
<b>ted_GoHome</b>	Moves to the start of the row.

<b>ted_GotoCursor</b>	Goes to a text cursor offset.
<b>ted_GoTop</b>	Moves to the top of the sed text.
<b>ted_GotoPosition</b>	Goes to a text position in the sed.
<b>ted_LeftChar</b>	Moves left one character.
<b>ted_LeftWord</b>	Moves left one word.
<b>ted_PageDown</b>	Moves down one page.
<b>ted_PageUp</b>	Moves up one page.
<b>ted_RightChar</b>	Moves right one character.
<b>ted_RightWord</b>	Moves right one word.
<b>ted_Search</b>	Searches for a string.
<b>ted_StartWorking</b>	Starts up a text editor.
<b>ted_UpChar</b>	Moves up one character.

## 2.17 Ted block functions

<b>ted_BlockAttr</b>	Sets the marked text block's attribute.
<b>ted_BlockCopy</b>	Copies a marked text block to buffer.
<b>ted_BlockCut</b>	Cuts a marked text block to buffer.
<b>ted_BlockDelete</b>	Deletes a marked text block.
<b>ted_BlockPaste</b>	Pastes buffer into a text block.

## 2.18 Ted I/O functions

<b>sed_ClearTB</b>	Resets the sed's text buffer.
<b>sed_GetTB</b>	Gets the sed's text buffer.
<b>sed_RewindTB</b>	Rewinds the sed's text buffer.
<b>sed_SetTB</b>	Sets the sed's text buffer.
<b>ted_ReadFile</b>	Fills a text buffer from a file.
<b>ted_WriteFile</b>	Writes a text buffer to a file.



## 2.19 Screen file functions

<b>sfile_Close</b>	Closes a screen file.
<b>sfile_LoadSed</b>	Loads a sed from a screen file.
<b>sfile_Open</b>	Opens a screen file.
<b>sfile_SaveSed</b>	Saves a sed to a screen file.

## 2.20 Help functions

<b>help_Close</b>	Closes the help system.
<b>help_GetChapter</b>	Gets the help's current chapter number.
<b>help_GetData</b>	Gets the help's data pointer.
<b>help_GetMessage</b>	Gets the help's current message number.
<b>help_GetParagraph</b>	Gets the help's current paragraph number.
<b>help_GetSize</b>	Gets the help's size parameter.
<b>help_GetText</b>	Gets the help's current text string.
<b>help_GetTitle</b>	Gets the help's current title string.
<b>help_Index</b>	Computes a help index.
<b>help_Init</b>	Initializes the help system.
<b>help_LookUp</b>	Looks up a message in the help system.
<b>help_Show</b>	Displays the appropriate help message.
<b>help_View</b>	Displays help messages in a scrolling box.
<b>help_Xref</b>	Displays cross-referenced help messages.

## 2.21 Keyboard Functions

<b>ascii</b>	Returns ASCII portion of a scancode.
<b>kb_Alt</b>	Checks if the Alt key is pressed.
<b>kb_CapsLock</b>	Checks if the keyboard is in Caps Lock mode.
<b>kb_Check</b>	Checks status of the keyboard buffer.
<b>kb_Clear</b>	Clears the keyboard buffer.

<b>kb_Control</b>	Checks if the Control key is pressed.
<b>kb_Idle</b>	Attaches a keyboard idle function.
<b>kb_Insert</b>	Checks if the keyboard is in Insert mode.
<b>kb_NumLock</b>	Checks if the keyboard is in Num Lock mode.
<b>kb_Read</b>	Reads a character from the keyboard.
<b>kb_Record</b>	Records/plays keystrokes to/from a file.
<b>kb_ScrollLock</b>	Checks if the keyboard is in Scroll Lock mode.
<b>kb_Shift</b>	Checks if the Shift key is pressed.

## 2.22 Memory allocation functions

<b>ocalloc</b>	Allocates and clears a block of memory.
<b>ofree</b>	Deallocates a memory block.
<b>omalloc</b>	Allocates a block of memory.
<b>orealloc</b>	Reallocates a block of memory.

## Appendix A: Function Reference

---

**Synopsis**

```
int ascii(scancode);  
    int scancode;          the scancode
```

**Description**

This routine strips off the higher byte of a scancode and returns the ASCII value of the scancode (the lower byte).

**Return Value**

Returns the ASCII portion of the scancode.

**See Also****kb\_Read****Example**

```
{  
    int scancode, key;  
  
    scancode = kb_Read();  
  
    key = ascii(scancode);  
    if (isprint(key)) {  
        sed_Overwrite(sed, key);  
    }  
}
```

**Synopsis**

```
void aux_func(name);
```

```
char *name;           auxiliary function to be prototyped
```

**Description**

This routine takes the name of an auxiliary function that you wish to prototype and performs that action.

**Return Value**

This routine has no return value.

**Example**

```
aux_func(aux_Mongoose);
```

```
main() {
```

```
/* ... */
```

```
aux_Mongoose(sed_DleTheScore, SED_POSTFEXIT, NULL, NULL);
```

```
}
```

**Synopsis**

```
sed_type bob_GetOwner(bob, fldnop);  
    sed_type bob;           the bobbed sed  
    int *fldnop;            pointer to field number
```

**Description**

This routine returns the sed that owns the given bob. It finds the field in the parent sed to which the bob is attached, and places the field number of this field into the integer pointed to by *fldnop*. If the bob is not attached to a sed, the value of the variable to which *fldnop* points is unchanged.

**Return Value**

Returns a pointer to the sed that owns the bob; NULL, if the bob is not attached to a sed.

**See Also**

**sed\_CreateBob, sed\_GetBob, sed\_GetAncestor**

**Example**

```
sed_type oursed, owner;  
bob_type bob;  
int fld;  
  
/* ... See if our sed is attached to another */  
  
bob = sed_GetBob(oursed);  
  
if (bob != NULL) {  
    owner = bob_GetOwner(bob, &fld);  
  
    /* oursed is attached to field 'fld' within sed 'owner' */
```

## bob\_GetPosition

Get a bob object's position

---

### Synopsis

```
void bob_GetPosition(bob, row, col);  
  
    bob_type bob;           the bob object  
    int *row;               row position of the bob  
    int *col;               column position of the bob
```

### Description

This routine returns the position of the upper left hand corner of the bob object into the locations pointed to by *row* and *col*. It gives the position in character coordinates.

### Return Value

There is no return value.

### See Also

### bob\_SetPosition

### Example

```
int row, col;  
  
bob_GetPosition(bob, &row, &col);
```

**Synopsis**

```
sed_type bob_GetSed(bob);  
    bob_type bob;           the bob object
```

**Description**

Returns the sed from which the bob was created. If the bob was created from an object other than a sed then **bob\_GetSed** returns NULL.

**Return Value**

Returns the sed from which the bob was created or NULL if the bob was created from an object other than a sed.

**Note**

This routine is implemented as a macro.

**See Also****sed\_CreateBob****Example**

```
bob = sed_CreateBob(mysed, BOB_DEPENDENT);  
  
/* ... */  
  
sed = bob_GetSed(bob);
```



### Synopsis

```
void bob_GetSize(bob, height, width);  
  
    bob_type bob;           the bob object  
    int *height;           the height of the bob  
    int *width;            the width of the bob
```

### Description

This routine returns the size of the bob object in the locations pointed to by *height* and *width*. It gives the size in character units.

### Return Value

There is no return value.

### See Also

#### bob\_GetPosition

### Example

```
int hgt, wid;  
  
bob_GetSize(bob, &hgt, &wid);
```

**Synopsis**

```
int bob_Go(bob);  
    bob_type bob;           the bob object
```

**Description**

This routine activates a bob object, which passes program control to it. For example, if a bob is created from a sed, then calling **bob\_Go** has the same effect as calling **sed\_Go**.

**Return Value**

Returns the value returned by the specific bob class used. For example, if the bob was created from a sed then the return value is the same as would have been returned by **sed\_Go**.

**See Also**

**sed\_CreateBob, sed\_Go**

**Example**

```
void bob_fkey(sed)  
    sed_type sed;  
{  
    bob_type bob;  
  
    bob = sed_GetFieldBob(sed, sed_GetFieldNo(sed));  
  
    switch( bob_Go(bob) ) {  
  
        /* ... */  
    }
```

**Synopsis**

```
boolean bob_IsSed(bob);  
bob_type bob;           the bob to be checked
```

**Description**

This routine checks if the bob has been created from a sed.

**Return Value**

Returns TRUE if the bob was created from a sed; FALSE, otherwise.

**Example**

```
if sed_IsBob(bob) {  
    menu = sed_GetMenu(bob_GetSed(bob));  
    menu_Printf(menu, "@f[#####]", beer_brand, &beer_funcs);  
    /* using a custom field function */  
}
```

### Synopsis

```
void bob_Pop(bob);  
    bob_type bob;           the bob object
```

### Description

This routines removes a bob's image from the display by firing its window. The area underneath the bob is repainted by windows that were obscured by it.

The bob and its window are not destroyed. To replace the bob onto the display call **bob\_Repaint**.

### Return Value

There is no return value.

### See Also

**bob\_Repaint**

### Note

This routine is implemented as a macro.

### Example

```
bob_Repaint(bob);           /* display the bob */  
ret = bob_Go(bob);  
bob_Pop(bob);               /* remove the bob */
```

### Synopsis

```
void bob_Repaint(bob);  
    bob_type bob;           the bob object
```

### Description

This routine repaints the bob object to the display. Using this routine on a bob created from a sed has the same affect as calling **sed\_Repaint**.

**bob\_Repaint** hires the bob's window if the window it is not currently employed.

### Return Value

There is no return value.

### See Also

**bob\_Pop**

### Note

This routine is implemented as a macro.

### Example


```
bob_Repaint(bob);           /* display the bob */  
ret = bob_Go(bob);  
bob_Pop(bob);               /* remove the bob */
```

### Synopsis

```
void bob_SetDepend(bob, depend);  
  
    bob_type    bob;           basic object  
    boolean     depend;       BOB_DEPENDENT or BOB_INDEPENDENT
```

### Description

This routine sets the dependency flag of *bob* to *depend*. *depend* has the value of either BOB\_DEPENDENT or BOB\_INDEPENDENT.

The principal difference between dependent and independent bobs is that dependent bobs are automatically painted when their parent is painted and clipped within the edges of their parent window. By contrast, independent bobs must be painted explicitly and can appear outside of their parent's edges. When a bob is embedded in a sed window by means of a **bob\_funcs** field, control passes automatically into a dependent bob; independent bobs act as popups that appear when a user presses  or clicks the mouse on the **bob\_funcs** field.

### Return Value

There is no return value.

### **Synopsis**

```
void bob_SetPosition(bob, row, col);  
  
    bob_type bob;           the bob object  
    int row;                row position of the bob  
    int col;                column position of the bob
```

### **Description**

This routine sets the position of the upper-left corner of the bob. The position is in character coordinates.

If the bob's window is employed, it moves on the display to the new position.

### **Return Value**

There is no return value.

### **See Also**

#### **bob\_GetPosition**

### **Note**

This routine is implemented as a macro.

### **Example**

```
int row, col;  
  
bob_GetPosition(bob, &row, &col);  
  
bob_SetPosition(bob, row + 5, col + 5);
```

**Synopsis**

```
void cls();
```

**Description**

This routine clears the background window. If there are other windows visible on the display they are not affected.

**Return Value**

There is no return value.

**Note**

This routine is implemented as a macro.

**Example**

```
void main()
{
    disp_Init(def_ModeGraphics, grwin_Class);
    cls();

    /* ... */

    disp_Close();
}
```



### Synopsis

```
#include "framer.h"

int frame_Go(frame, ch, data);

    sed_type frame;          the frame object
    int ch;                  starting character
    VOID *data;              frame data pointer
```

### Description

This routine activates a framer menuing system. It searches for the first choice starting with the letter *ch*. If no choice starts with *ch* then 0 is returned. To ignore the first letter search, set *ch* equal to the space character, ' '. *data* is a data pointer that passes data to the user-supplied functions.

Consult the *C-scape Manual* for a detailed description of the framer menuing system.

### Return Value

Returns the value returned by the user function, the value specified in the definition structure, or 0 if **Esc** was pressed.

### See Also

**frame\_Open**

### Example

```
#include "framer.h"

void main()
{
    sed_type frame;
    /* ... */
    frame = frame_Open(frame_list, bd_cua, 0x07, 0x70, 0x07);

    sed_Repaint(frame);
    frame_Go(frame, ' ', (VOID *) my_data);
}
```

### Synopsis

```
#include "framer.h"
```

```
boolean frame_Lock(frame, family, member, attribute);
```

sed_type frame;	the frame object
int family;	pulldown menu affected
int member;	choice within pulldown
byte attribute;	attribute of locked choice

### Description

This routine “locks” a menu choice within the frame menuing system. A locked menu choice appears but cannot be selected by the user. The menu choice is selected by *family* and *member*. *family* selects the pulldown menu (starting with 0) and *member* selects the choice within the pulldown (starting with 0). The attribute of the choice is changed to *attribute*.

### Return Value

Returns FALSE if an invalid choice is specified, TRUE otherwise.

### See Also

**frame\_UnLock**

### Example

```
/* ... */  
  
frame_Lock(frame, 0, 2, 0x07);
```

### Synopsis

```
#include "framer.h"
```

```
sed_type frame_Open(frame_list, border, bck_attr, sel_attr,  
bd_attr);
```

struct frame_def frame_list[];	the frame definition structure
bd_fptr border;	the border function
byte bck_attr;	the background attr
byte sel_attr;	the selected menu choice attr
byte bd_attr;	the border attr

### Description

This routine creates a frame menu system from a frame definition structure, *frame\_list*. *border* is the border attached to the pulldown menus. *bck\_attr*, *sel\_attr*, and *bd\_attr* determine the colors of the frame menu.

Consult the *C-scape Manual* for a detailed description of the frame menuing system.

### Return Value

Returns a pointer to the new frame object. A NULL pointer values indicates insufficient memory or an error in the definition structure.

### See Also

**frame\_Go, sed\_Close**

### Note

User functions for slug and framer menuing systems and also for bobs all have the same format and may be used interchangeably. User functions, however, are not a recommended part of the C-scape library; see Section 16.3 of the *C-scape Manual* for more information.

Also, this routine is implemented as a macro.

## Example

```
#include "framer.h"

/* use macro to prototype user functions */
ufunc_func(save_file);
ufunc_func(load_file);
ufunc_func(fmt_text);
ufunc_func(fmt_block);

struct frame_def main_menu[] = {
    { "File", NULL, 0 },
    { "Save", save_file, 0 },
    { "Load", load_file, 0 },
    { FRAME_END },
    { "Format", NULL, 0 },
    { "Text", fmt_text, 0 },
    { "Block", fmt_block, 0 },
    { FRAME_END },
    { FRAME_END }
};

void main()
{
    sed_type frame;

    /* ... */

    frame = frame_Open(main_menu, FNULL, 0x07, 0x70, 0x07);

    sed_Repaint(frame);
    frame_Go(frame, ' ', (VOID *) my_data);

    /* ... */

    sed_Close(frame);
}

int fmt_block(sdata, idata)
    VOID *sdata; /* from frame_Go */
    int idata; /* from the frame_def */
{
    /* code to format a block */

    return(1);
}
```

### Synopsis

```
#include "framer.h"

boolean frame_UnLock(frame, family, member);

    sed_type frame;          the frame object
    int family;              pulldown menu affected
    int member;              choice within pulldown
```

### Description

This routine “unlocks” a menu choice within the frame menu; it counteracts the effects of **frame\_Lock**. It uses *family* and *member* to select a menu choice. *family* selects the pulldown menu (starting with 0) and *member* selects the choice within the pulldown (starting with 0).

### Return Value

Returns FALSE if an invalid choice is specified, TRUE otherwise.

### Note

This routine is implemented as a macro.

### See Also

**frame\_Lock**

### Example

```
/* ... */

frame_UnLock(frame, 0, 2);
```

### Synopsis

```
#include "helpdecl.h"
```

```
void help_Close();
```

### Description

Frees the space that the help system uses. You cannot use the help system after you close it. You can re-open it, however, with **help\_Init**.

### Return Value

There is no return value.

### See Also

#### help\_Init

### Example

```
void main()
{
    FILE *fp;

    /* initialize display interface */
    disp_Init(def_ModeGraphics, NULL);

    fp = fopen("file.hlp", "rb");
    if (fp == NULL) {
        disp_Close();
        printf("Unable to open help file\n");
        return;
    }

    help_Init(fp, help_View, 6000, NULL);

    /* ... */
    help_Close();
    fclose(fp);

    disp_Close();
}
```

### Synopsis

```
#include "helpdecl.h"

int help_GetChapter(help);
    help_type help;          the help object
```

### Description

This routine returns the current help chapter number. **help\_Show** sets the current chapter number.

This routine can only be used within a help display function.

### Return Value

Returns the current chapter number.

### See Also

**help\_GetParagraph, help\_Show**

### Note

This routine is implemented as a macro.

### Example

```
int chapter;

chapter = help_GetChapter(help);
```

### Synopsis

```
#include "helpdecl.h"

VOID *help_GetData(help);
    help_type help;          the help object
```

### Description

This routine returns the generic data pointer for the help object. The **help\_Init** function sets this pointer. The data pointer passes data to the help display function.

This routine can only be used within a help display function.

### Return Value

Returns the help data pointer.

### See Also

#### help\_Init

### Note

This routine is implemented as a macro.

### Example

```
struct hv_struct *hv;

/* Was an hv_struct passed to help_Init? (from help_View) */
hv = (struct hv_struct *) help_GetData(h);

if (hv != NULL) {

    /* use contents of structure */

} else {

    /* use defaults */

}
```



## help\_GetMessage

Get the help's current message number

---

### Synopsis

```
#include "helpdecl.h"

int help_GetMessage(help);
    help_type help;        the help object
```

### Description

This routine returns the current help message number. **help\_Show** sets the current message number by looking up the current chapter and paragraph in the help file index.

This routine can only be used within a help display function.

### Return Value

Returns the current help message number.

### See Also

**help\_Show**

### Note

This routine is implemented as a macro.

### Example

```
int msg;

msg = help_GetMessage(help);
```

## help\_GetParagraph

Get the help's current paragraph number

---

### Synopsis

```
#include "helpdecl.h"

int help_GetParagraph(help);
    help_type help;          the help object
```

### Description

This routine returns the current help paragraph number. **help\_Show** sets the current paragraph number. If no help message exists for a chapter, **help\_Show** sets the current paragraph number to 0.

This routine can only be used within a help display function.

### Return Value

Returns the current help paragraph number.

### See Also

**help\_GetChapter**, **help\_Show**

### Note

This routine is implemented as a macro.

### Example

```
int paragraph;

paragraph = help_GetParagraph(help);
```

### Synopsis

```
#include "helpdecl.h"

unsigned int help_GetSize(help);
    help_type help;          the help object
```

### Description

This routine returns the size parameter for the help object. **help\_Init** sets the size parameter; this parameter represents the size of the buffer allocated to hold the text of the help messages. This value limits the maximum size of help messages.

This routine can only be used within a help display function.

### Return Value

Returns the help size parameter.

### See Also

**help\_Init**

### Note

This routine is implemented as a macro.

### Example

```
unsigned int size;

size = help_GetSize(help);
```

### Synopsis

```
#include "helpdecl.h"

char *help_GetText(help);
    help_type help;          the help object
```

### Description

This routine returns the current help text string. **help\_Show** extracts the current help text string from the help file. The text string is a '\0' terminated string with '\n's at the end of each line. The help display function usually displays the text string.

This routine can only be used within a help display function.

### Return Value

Returns the current help text string.

### See Also

**help\_Show**

### Note

This routine is implemented as a macro.

### Example

```
/* from help_View ... */

opc_View(NULL, help_GetTitle(h), 0, help_GetText(h));

/* ... */
```

### Synopsis

```
#include "helpdecl.h"

char *help_GetTitle(help);
    help_type help;          the help object
```

### Description

This routine returns the current help title string. **help\_Show** extracts the current help title string from the help file. The help display function usually displays the title string.

This routine can only be used within a help display function.

### Note

This routine is implemented as a macro.

### Return Value

Returns the current help title string.

### See Also

#### **help\_Show**

### Example

```
/* from help_View ... */ wait for mike...

opc_View(NULL, help_GetTitle(h), 0, help_GetText(h));

/* ... */
```

## Synopsis

```
#include "helpdecl.h"

int help_Index(chapter, paragraph);
    int chapter;           the help chapter
    int paragraph;         the help paragraph
```

## Description

This routine looks up in the help index the message number associated with *chapter* and *paragraph*. If no message exists for *paragraph*, the message for paragraph 0 is looked up. If no message is found, -1 is returned. The help object's chapter and paragraph parameters are updated.

## Return Value

Returns the message number or (-1) if unable to find a message.

## See Also

## help\_Init

## Example

```
int success;

success = help_LookUp(2, 3);
```

### Synopsis

```
#include "helpdecl.h"

int help_Init(fp, disp, size, data);

FILE *fp;           the help file pointer
int (*disp)();       the help display function
unsigned int size;   size of the message buffer
VOID *data;          the help data pointer.
```

### Description

This routine initializes the help system. *fp* must point to a valid, open help file. You must open the help file for reading in the binary mode. It is a good idea to expand all the tabs in the help file to spaces as most display functions do not translate tabs to spaces.

*disp* must be the address of a display function. **help\_Show** calls the display function whenever it displays a help message. You can write custom functions to display help messages in the same manner as border and field functions.

*size* is used to determine the size, in bytes, of the help message buffer. The message buffer holds a message after it is read in from the help file. *size* determines the maximum size of a help message (characters and spaces); it should be set large enough to display the largest help message but not so large that space will be wasted.

The help's data pointer is set to *data*. The data pointer is used to pass data to the help data function.

This routine parses the help file and allocates storage to hold the help file indexing information. **help\_Show** uses this information to look up help messages in the help file. The help file has three sections separated by %% at the beginning of a line.

The first section of the help file is reserved for comments. The second section of the file contains the help index. The third section of the file contains the help messages.

The help messages are addressed by two indices, chapter and paragraph. The help index maps chapters and paragraphs to message numbers. The index syntax is “chapter,paragraph:message”. The chapter number must be in the first column and the index entries must be listed in increasing order. The first chapter is 1. The first paragraph is 0. Try to keep the numbers small (i.e., use chapter numbers 1, 2, 3, 4 not 1, 10, 23, 56) in order to save space.

Messages are designated by ‘.’ followed by a message number. The ‘.’ must be in the first column and messages must be listed in increasing order. The first line of each message is treated as a title string. The last message should be terminated by a ‘.’ at the beginning of the next line. The first message is 1. As with chapters, try to keep the numbers small (i.e., use message numbers 1, 2, 3, 4 not 1, 10, 23, 56) in order to save space.

You can place a ‘.’ followed by a ‘!’ at the start of a line to comment out a line in a help message.

When **help\_Show** is called, it looks up a help message with chapter and paragraph. If an index exists for the chapter and paragraph then the appropriate message is displayed. If not, an attempt is made to find a message for the same chapter number and paragraph 0. If a message exists for paragraph 0 of the chapter, **help\_Show** displays it, otherwise it displays no message.

#### Sample Help File

```
%%  
1,1:1  
1,2:2  
1,3:3  
1,4:4  
%%  
.1  
Help Message One  
This is a help message.  
.! This line is commented out  
.!  
.2  
Help Message Two  
This is a two  
line help message.  
.3
```



Help Message Three

Enter the correct time into this field.

.4

Help Message Four

Type your name into the appropriate field.

Use the arrow keys to move the cursor.

.End

An error value is returned if **help\_Init** is unsuccessful.

## Return Value

Returns one of the following values:

HELP_OK	The help system has been successfully initialized.
HELP_NOMEM	<b>help_Init</b> could not allocate enough memory to initialize the help system.
HELP_BADARG	<b>help_Init</b> received bad arguments. (Either <i>fp</i> or <i>disp</i> is equal to NULL or help has already been initialized.)
HELP_BADFILE	<b>help_Init</b> found an error or unexpected end in the help file.

## See Also

**help\_Close**, **help\_Show**

## Example

```
void main()
{
    FILE *fp;

    /* initialize display interface */
    disp_Init(def_ModeGraphics, NULL);

    fp = fopen("file.hlp", "rb");

    if (fp == NULL) {
        disp_Close();
        printf("Unable to open help file\n");
        return;
    }

    help_Init(fp, help_View, 6000, NULL);
}
```

# help\_LookUp

Look up a message in the help system

---

## Synopsis

```
#include "helpdecl.h"

boolean help_LookUp(msg);

    int msg;                the message number
```

## Description

This routine looks up a message in the help system. It adjusts the help system's parameters (*text*, *title*, etc.).

## Return Value

Returns TRUE if successful.

## See Also

**help\_Index**, **help\_Init**

## Example

```
/* ... */

if (msg > 0) {
    /* get new help message */

    if (!help_LookUp(msg)) {
        return(0);
    }
}

/* ... */
```

## help\_Show

Display the appropriate help message

---

### Synopsis

```
#include "helpdecl.h"

int help_Show(chapter, paragraph);
    int chapter;           the help chapter
    int paragraph;        the help paragraph
```

### Description

This routine searches for the appropriate help message in the help file. If it finds a message, it calls the help display function. If the specified paragraph does not have a help message, it tries paragraph 0.

There is no help for chapter 0.

### Return Value

Returns 1 if the message was displayed, 0 otherwise.

### See Also

**help\_Init, sed\_SetLabel**

### Example

```
/* from special_key */

#include "helpdecl.h"

/* ... */

switch (scancode) {
case FN1:
    help_Show(sed_GetLabel(sed), sed_GetFieldNo(sed) + 1);
    return(TRUE);

/* ... */
```

### Synopsis

```
#include "helpdecl.h"

int help_View(help);

    help_type help;          the help object
```

### Description

This routine is a help display function. It displays help messages. It is attached to the help system with the **help\_Init** function.

This routine uses the **opc\_View** routine to display the help message. You can set the parameters for **opc\_View** by pointing the help data pointer to an **hv\_struct** containing the data. The **hv\_struct** is listed below:

```
struct hv_struct {
    int    row;              /* row position of help window */
    int    col;              /* col position of help window */
    int    height;           /* height of help window */
    int    width;            /* width of help window */
    byte   color;            /* color of help window */
    bd_fptr border;          /* help window border function */
};
```

Refer to the Help chapter of the *C-scape Manual* for more information on **help\_View**.

### Return Value

Returns 1 if the message was displayed, 0 otherwise.

### See Also

**help\_Init**, **help\_Show**, **opc\_View**

## Example

```
struct hv_struct hvd = { -1, -1, -1, -1, 0x70, NULL};

void main()
{
    FILE *fp;

    /* initialize display interface */
    disp_Init(def_ModeGraphics, NULL);

    fp = fopen("file.hlp", "rb");
    if (fp == NULL) {
        disp_Close();
        printf("Unable to find help file\n");
        return;
    }

    help_Init(fp, help_View, 6000, (VOID *) &hvd);
    /* ... */
}
```


### Synopsis

```
#include "helpdecl.h"

int help_Xref(help);
    help_type help;          the help object
```

### Description

This routine is a help display function. It displays help messages. It is attached to the help system with the **help\_Init** function.

This routine supports a cross-referenced help system using data in the help text file to determine the cross-references. Specifically, key words in a help message are delimited by: "@3[key word]". In this case the phrase "key word" will be highlighted whenever the appropriate help message appears. The "3" indicates that help message number 3 should be displayed if the user selects "key word". The user can cycle through as many help messages as are cross-referenced. The user can use the  key to step back through previously displayed help screens.

You can set the display parameters by pointing the help data pointer to an **hx\_struct** containing the data. The **hx\_struct** is listed below:

```
struct hx_struct {
    byte bk_clr;          /* background color of help window */
    byte reg_clr;         /* color of choices */
    byte sel_clr;         /* color of selected choice */
    byte bd_clr;          /* color of help window border */
    bd_fptr border;       /* help window border function */
};
```

Refer to the Help chapter of the *C-scape Manual* for more information on **help\_Xref**.

The border function, **bd\_xref**, is usually used with this help display function.

### Return Value

Returns 1 if the message was displayed, 0 otherwise.

## See Also

**help\_Init, help\_Show**

## Example

```
struct hx_struct hxd = { 0x07, 0x0f, 0x70, 0x07, bd_xref };

void main()
{
    FILE *fp;

    /* initialize display interface */
    disp_Init(def_ModeGraphics, NULL);

    fp = fopen("file.hlp", "rb");
    if (fp == NULL) {
        disp_Close();
        printf("Unable to help file\n");
        return;
    }

    help_Init(fp, help_Xref, 6000, (VOID *) &hxd);
    /* ... */
}
```

### Synopsis

```
#include "scancode.h"

boolean inter_field(sed, scancode);

sed_type sed;           the sed
int scancode;           keystroke to process
```

### Description

This routine handles the inter-field movement for the standard field functions in the following manner:

UP	Call <b>sed_DecField</b>
DOWN	Call <b>sed_IncField</b>
TAB	Call <b>sed_LeftField</b> .
SHFT_TAB	Call <b>sed_RightField</b> .
ESCAPE	Exit the sed, set the baton to 0.
ENTER	Call <b>sed_IncField</b> . If called from the last field, exit the sed and set the baton equal to the current field number plus one.

It is possible to replace **inter\_field** with a modified version.

**inter\_field** can be used as a sed's special function.

### Return Value

Returns TRUE if a keystroke was processed.

### See Also

**inter\_field\_grid**, **inter\_page**, **sed\_SetSpecial**



## Example

```
void string_fkey(sed)
    sed_type sed;
{
    int scancode;

    scancode = kb_Read();
    if (sed_DoSpecial(sed, scancode))
        return;
    if (special_key(sed, scancode))
        return;
    if (inter_field(sed, scancode))
        return;
    if (inter_page(sed, scancode))
        return;
    /* ... */
}
```

### Synopsis

```
#include "scancode.h"

boolean inter_field_grid(sed, scancode);

    sed_type sed;           the sed
    int scancode;          keystroke to process
```

### Description

This routine handles the orthogonal inter-field movement for some of the standard field functions in the following manner:

UP	Call <b>sed_UpField</b>
DOWN	Call <b>sed_DownField</b>
RIGHT	Call <b>sed_RightField</b>
LEFT	Call <b>sed_LeftField</b>
ESCAPE	Exit the sed, set the baton to 0.
ENTER	Exit the sed, set the baton equal to the current field number plus one.

It is possible to replace **inter\_field\_grid** with a modified version.

**inter\_field\_grid** can be used as a sed special function.

### Return Value

Returns TRUE if a keystroke was processed.

### See Also

**inter\_field**, **inter\_page**, **sed\_SetSpecial**

## Example

```
void gmenu_fkey(sed)
    sed_type sed;
{
    int scancode;

    scancode = kb_Read();

    if (scancode == ENTER) {
        sed_SetBaton(sed, sed_GetFieldNo(sed)+1);
        sed_ToggleExit(sed);
        return;
    }

    if (sed_DoSpecial(sed, scancode))
        return;
    if (special_key(sed, scancode))
        return;
    if (inter_field_grid(sed, scancode))
        return;
    if (inter_page(sed, scancode))
        return;

    /* ... */
}
```

### Synopsis

```
#include "scancode.h"

boolean inter_page(sed, scancode);
    sed_type sed;           the sed
    int scancode;           keystroke to process
```

### Description

The standard field functions use this routine to handle inter-page movement as follows:

PGUP Call **sed\_PageUp**

PGDN Call **sed\_PageDown**

You can replace this routine to change the way the standard field functions handle inter-page movement.

### Return Value

Returns TRUE if a keystroke was processed.

### Example

```
void string_fkey(sed)
    sed_type sed;
{
    int scancode;

    scancode = kb_Read();
    if (special_key(sed, scancode))
        return;
    if (inter_field(sed, scancode))
        return;
    if (inter_page(sed, scancode))
        return;
    /* ... */
}
```

**Synopsis**

```
char invert_char(c);  
    char c;                the character to invert.
```

**Description**

This routine swaps the high and low nibbles (4 bits) of a character. It is useful for computing the inverse of a color attribute.

**Return Value**

Returns the inverted character.

**Example**

```
char color, inverse_color;  
  
/* ... */  
  
inverse_color = invert_char(color);
```

### Synopsis

```
void menu_Destroy(menu);  
    menu_type  menu;      the menu
```

### Description

This routine closes a menu and frees any storage it used. You cannot use a menu after you have closed it. You can only use this routine when you have not created a sed from the menu. If you have created a sed from the menu, **sed\_Close** automatically closes the menu object.

### Return Value

There is no return value.

### See Also

**menu\_Open**, **sed\_Close**

### Example

```
/* ... */  
  
if ((sed = sed_Open(menu)) == NULL) {  
    menu_Destroy(menu);  
    return(FALSE);  
}
```

### Synopsis

```
void menu_Flush(menu);  
    menu_type menu;    the menu
```

### Description

This routine releases storage used by the **menu\_Printf** parsing engine. It is usually called after the last of a series of calls to **menu\_Printf**.

When **menu\_Printf** is first called, it allocates memory for parsing format strings. Once a menu has been completely defined, this memory is no longer needed. Calling **menu\_Flush** frees this memory for use by others. If you call **menu\_Printf** after flushing a menu, it will allocate new memory.

Calling **menu\_Flush** is optional. It is useful when you think that you will not call any further **menu\_Printfs** on a menu.

When **sed\_Close** closes the menu, it will flush the menu if **menu\_Flush** has not been called.

### Return Value

There is no return value.

### See Also

**menu\_Printf**

### Example

```
menu = menu_Open();  
  
menu_Printf(menu, "Am I having fun yet?");  
  
menu_Flush(menu);  
  
/* ... */
```

### Synopsis

```
int menu_GetCol(menu);  
    menu_type menu;      the menu
```

### Description

This routine returns the current column number of the menu. This is the column position at which the next **menu\_Printf** will write a character unless a “@p[]” or “\n” is encountered.

### Return Value

Returns the current menu column.

### See Also

**menu\_GetRow**, **menu\_Printf**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
menu_Printf(menu, "@p[%d]X marks the spot", menu_GetCol(menu)+5);
```



### Synopsis

```
int menu_GetFieldCol(menu, fieldno);  
  
menu_type menu;           the sed holding the field  
int field;                the field in question
```

### Description

This routine returns the integer value of the leftmost position of a field.

### Return Value

Returns the leftmost position of the field.

### Note

This routine is implemented as a macro.

### Example

```
int row, column;  
boolean var;  
  
row = menu_GetFieldRow(sed, fieldno);  
column = menu_GetFieldCol(sed, fieldno);  
menu_Printf(sed_GetMenu(sed), "@p[%d, %d]@f[###]", row, column +  
15,  
    &var, &yesno_funcs);
```

### Synopsis

```
int menu_GetFieldRow(menu, fieldno);  
  
menu_type menu;      the menu holding the field  
int field;           the field in question
```

### Description

This routine returns the integer value of the row of a field.

### Return Value

Returns the row of the field.

### Note

This routine is implemented as a macro.

### Example

```
int row, column;  
boolean var;  
  
row = menu_GetFieldRow(sed, fieldno);  
column = menu_GetFieldCol(sed, fieldno);  
menu_Printf(sed_GetMenu(sed), "@p[%d, %d]@f[###]", row, column +  
15,  
    &var, &yesno_funcs);
```

### Synopsis

```
int menu_GetHeight(menu);  
    menu_type menu;        the menu
```

### Description

This routine returns the current number of rows in the menu.

### Return Value

Returns the current number of rows in the menu.

### See Also

**menu\_GetWidth, menu\_Printf**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
menu_Printf(menu, "@p[%d,0]That row", menu_GetHeight(menu) + 3);
```

### Synopsis

```
int menu_GetRow(menu);  
    menu_type menu;    the menu
```

### Description

This routine returns the current row number of the menu. This is the row position at which the next **menu\_Printf** will write a character unless a “@p[]” or “\n” is encountered.

### Return Value

Returns the current menu row.

### See Also

**menu\_GetCol**, **menu\_Printf**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
menu_Printf(menu, "@p[%d,0]This row", menu_GetRow(menu) + 3);
```

### Synopsis

```
int menu_GetWidth(menu);  
    menu_type  menu;      the menu
```

### Description

This routine returns the current number of columns in the menu.

### Return Value

Returns the current number of columns in the menu.

### See Also

**menu\_GetHeight, menu\_Printf**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
menu_Printf(menu, "@p[%d]Centered", menu_GetWidth(menu)/2 - 4);
```

## Synopsis

```
menu_type menu_Open();
```

## Description

This routine creates a new menu object and returns a handle to it. If there is not enough memory to open a new menu object, the routine returns NULL.

The menu object can be considered as the “blueprint” from which a screen is created. It contains all the static details of the screen such as field and text positions. A menu object is built using a series of calls to **menu\_Printf**. Once a menu has been created, you can create a sed from it with **sed\_Open**. The sed contains all the dynamic information of a screen such as positioning, field contents, etc. When you close the sed with **sed\_Close**, the menu object is automatically closed.

## Return Value

Returns a handle to the new menu object. A NULL pointer value indicates insufficient memory.

## See Also

**menu\_Destroy**, **sed\_Close**

## Example

```
menu_type menu;
sed_type sed;

menu = menu_Open();
menu_Printf(menu, "Your menu here");
menu_Flush(menu);

sed = sed_Open(menu);

/* ... */
sed_Close(sed);
```

## Synopsis

```
boolean menu_Printf(menu, fmt, arg1, arg2,...);

    menu_type menu;        the menu
    char *fmt;             the format string
```

## Description

This routine defines a menu object. It is analogous to C's **printf**. You can use as many **menu\_Printf**s as you need to define a menu.

The **menu\_Printf** format string contains the information that defines a menu object. There are three categories of characters in the format string: plain characters, conversion characters, and command characters. The plain characters simply go into the menu's text buffer. The conversion characters (**%** substitutions) grab arguments from the argument list and format them. The command characters (**@** commands) give specific commands to **menu\_Printf**. Some command characters grab arguments from the argument list.

It is very important that the argument list contains enough data for all the conversions and commands specified in the format string. You must also make sure that the arguments are of the correct type and that they are in the correct order. Unpredictable results will occur if these conditions are not met.

The format codes are listed below:

command	name	example	description
@a[	color	@a[7] @a[0x70]	Change current menu color to color [color]. <i>color</i> is an integer. If <i>color</i> starts with "0x" it is treated as a hexadecimal number.

command	name	example	description
@p[]	position	@p[2,4]	Set menu row and column to position [row, column]. <i>row</i> and <i>col</i> are ints.
		@p[6]	Set menu column to position within current menu row.
@f[]	field	@f[\$(###)]	Define a field. '#' signs denote "writeable" positions; all others are printed literally, and act as "nonwriteables".
		@fp[###]	Define a "protected" field.
		@fd[####]	Bind data to the field data pointer from the argument list.
		@fd3[###]	Define a field with 3 (or however many) data pointers and bind data to the pointers from the argument list.
		@fD3[####]	Define a field with 3 (or however many) data pointers without binding the data to the pointers from the argument list. The pointers remain uninitialized.
		@fh2[Exit]	Define a field with the second character (the 'x') highlighted.
		@fw2[###]	Define a field with width 3 and displayed width 2 (or any other value).
		@fb[]	Define a field with an attached bob object.
@[]	repeat	@f{ho}[##]	Define a field with the name "ho".
		@[3,xyz]	Repeat a string <i>count</i> times; @[count,string]. <i>count</i> is an int. <i>string</i> consists of all the chars between the comma ',' and the closing bracket ']'.
@	quote	@@, @#, @	Quote a <b>menu_Printf</b> special character.
\	backslash	\n	Used for C control characters.
%	percent	%d	All % substitutions are handled in the same manner as <b>printf</b> .
		%s	
		%6f	
%%		%%	Quote a '%' character.



C uses '%' as a control character to indicate a formatting command. **menu\_Printf** adds to this its own set of formatting commands which use '@' as a control command.

The syntax for specifying the printing position in a menu is "@p[5,10]": the control character "@" followed by the formatting code "p", and then the row, column position surrounded by brackets. In this case printing would begin at row 5, column 10.

If there is only one number within the brackets, such as "@p[22]", the printing position's row remains unchanged and the column is set to the new value. It is possible to use "%d" substitutions within a position command:

```
menu_Printf(menu, "@p[%d,%d]", row, column);
```

To create a field, use the syntax "@f[###]###-####": the "@" character followed by "f", and a field mask surrounded by brackets. The "#" character denotes a "writeable" character. Writeable characters are those over which the user can type. Anything else (such as the parentheses and dash in this example), is treated as a "nonwriteable" character. The user cannot type over it.

Whenever **menu\_Printf** encounters an "@f[]" sequence, it takes the next two arguments from the argument list and binds them to the newly created field. The first argument becomes the field variable and should be a pointer to the data type expected for the field; the second argument is the address of the `field_function` struct to be associated with the field. For instance:

```
menu_Printf(menu, "@f[####]", s, &string_funcs);
```

The preceding code defines a field four writeable positions long. The field variable is `s` and the field function is **string\_funcs**.

The "@f[]" sequence can contain certain modifying characters, such as 'p', 'd', 'b', 'w', and '{}'. The 'p' in "@fp[]" signifies that the field will be "protected." The 'd' in "@fd[]" signifies that data will be bound to the field's generic data pointer when the field is defined. The data pointer is taken from the argument list. The 'b' in "@fb[]" signifies that a bob object will be bound to the field. The bob object also comes from the argument list. The "w10" in "@fw10[]" signifies that the field will have a displayed width of 10. The "{toad}" in "@f{toad}" signifies that the field will have the name "toad". These modifying characters can be used together and in any order:

```
menu_Printf(menu, "@fpdw2[####]", s, &string_funcs, "abc");
```

The preceding code defines a protected field four writeable positions long. The field variable is *s* and the field function is **string\_funcs**. A pointer to the string "abc" is bound to the field's generic data pointer.

You can define a field with multiple data pointers. For example, you may wish to bind a string to the field so when users enter the field, a message appears; you may want another string to hold validation data for the field. The syntax for creating fields with multiple data pointers is "@fd*n*", where *n* is a digit. The data pointers are initialized by the succeeding arguments following the field function in the argument list:

```
menu_Printf(menu, "@fd2[####]", &val, &int_funcs,  
    "Enter an integer", "(1,22)");
```

The preceding code defines a field four writeable positions long with two data pointers associated with it. The field variable is *val* and the field function is **int\_funcs**. A pointer to the string "Enter an integer" is bound to the field's first data pointer. A pointer to the string "(1,22)" is bound to the field's second data pointer. An argument must be provided for each data pointer.

You can observe and modify the data pointers with the commands **sed\_GetFieldData** and **sed\_SetFieldData**.

To change the current background text printing color, use the syntax "@a[7]": the "@" character followed by a "a", and the color value surrounded by brackets. In this case, the color is set to the value of 7. If the color starts with "0x" it is treated as a hexadecimal number: "@a[0x70]" indicates a color value of 70 hex.

To repeat a string multiple times, use the syntax "@[3,abc]": the "@" character followed by an open bracket, a repeat count delimited by a comma followed by the string to be repeated. The string consists of all the characters between the comma and the closing bracket. The repeat syntax used above is equivalent to "abcabcabc".

"@" can also be used as a quote character. For example, to print an "@" to the menu, use "@@"; to print "#", use "@#"; and to print "]", use "@]".

The newline character ('\n') changes the printing position to the start of the next line. Other C control characters, such as backspace ('\b'), print out their MS DOS graphics representation.

Percent ('%') substitution can be done anywhere in the string. The syntax is identical to **printf**; “%s” means print a string, “%2d” means print two digits of a decimal int, etc. As with **printf**, there should be one argument for each percent in the string.

**menu\_Printf** does not support the '\*' character within percent substitutions (i.e., “%\*s” is not supported).

If **menu\_Printf** cannot allocate sufficient memory it returns FALSE. You can also test this with **oak\_GetErrno**.

### **Return Value**

Returns TRUE if successful and FALSE otherwise.

### **See Also**

**menu\_Flush**, **menu\_UnPrintf**, **sed\_ProtectField**, **sed\_SetFieldData**, **printf**

## Example

```
/* #includes and initializations here */

int test(name, phone, count)
    char *name;
    char *phone;
    long *count;
{
    menu_type menu;
    sed_type sed;
    int ret;

    menu = menu_Open();
    menu_Printf(menu, "@p[0,0]@a[7]This is a test....");
    menu_Printf(menu, "@p[3,0]Name:");
    menu_Printf(menu, "@p[3,7]@fd[#####]",
        name, &string_funcs, "Enter your name");

    menu_Printf(menu, "@p[6,0]Phone:");
    menu_Printf(menu, "@p[6,7]@f[(###) ###-####]",
        phone, &string_funcs);

    menu_Printf(menu, "@p[9,0]Count:");
    menu_Printf(menu, "@p[9,7]@f{count}[#####]",
        count, &long_funcs);

    menu_Flush(menu);

    sed = sed_Open(menu);
    sed_SetColors(sed, 0x07, 0x07, 0x70);
    sed_SetBorder(sed, bd_cua);
    sed_SetPosition(sed, 8, 19);

    sed_Repaint(sed);
    ret = sed_Go(sed);

    sed_Close(sed);
    return(ret);
}
```

### Synopsis

```
void menu_SetWrapWidth(menu, wwid);  
  
    menu_type menu;          the menu  
    int wwid;                word wrap width
```

### Description

This routine sets the word wrap width of the menu to *wwid*. The word wrap width is the maximum displayed width of any row in the menu. If any of the menu's rows are longer than *wwid* they are split at the appropriate word break. The default word wrap width of a menu is 32000.

### Return Value

There is no return value.

### Note

This routine is implemented as a macro.

### Example

```
menu_Printf(menu, "This sentence might be too long.");  
menu_Flush(menu);  
  
menu_SetWrapWidth(menu, 20);  
  
sed = sed_Open(menu);  
  
/* ... */
```

### Synopsis

```
char *menu_UnPrintf(menu, fieldno, buf, row, col, dflag)
```

menu_type menu;	the menu
int fieldno;	the field number
char *buf;	a buffer for the return value
int row;	new row of the field
int col;	new column of the field
int dflag	data pointer translation flag

### Description

This routine takes a field number and creates for it a format string, such as the one passed as the second argument to **menu\_Printf**. *row* and *col* take values for new row and column values for the field's position; if you give either a value of -1, **menu\_UnPrintf** uses the field's old row or column. The routine places the format string in the variable *buf*, which must be as long as the format string (including the "@p[row,col]@f[]", any other command codes, and the length of the actual field spec).

The *dflag* variable determines the handling of data pointers. If *dflag* is non-zero, "@fd" is translated to "@fD". If *dflag* is 0, **menu\_UnPrintf** generates an "@fd" string; if *dflag* is 1, the routine generates an "@fD" string; if *dflag* is 2, **menu\_UnPrintf** will generate the "@fd" string, though without the "d" if there is only one NULL data pointer. See the **menu\_Printf** entry in this manual for more information on the "d" and "D" options.

### Return Value

Returns the buffer in which it places the format string.

## Example

```
menu = sed_GetMenu(sed);
char buf[50], string_var[30];

/* duplicate the first field, a string funcs field */
menu_Printf(menu,
    menu_UnPrintf(menu, 0, buf, -1, menu_GetFieldCol(menu, 0) + 5,
1),
    string_var, &string_funcs);
```

## Description

These routines refer to an `opc_type`, as do the routines `opc_Edit`, `opc_View`, and so on. This construct allows you to manipulate a popup box's features as you would for a `sed` or other window. Aside from its various attributes and position/dimension parameters (see below), it contains the following elements:

```
#include <time.h>
#include "popdecl.h"

void opc_SetTitle(opc, title);
char *opc_GetTitle(opc);

    char *title;          title string

void opc_SetPrompt(opc, prompt);
char *opc_GetPrompt(opc);

    char *prompt;         prompt string

void opc_SetData(opc, data);
VOID *opc_GetData(opc);

    VOID *data;           generic data pointer

void opc_SetLabel(opc, label);
int opc_GetLabel(opc);

    int label;            help label

void opc_SetBorder(opc, border);
bd_fptr opc_GetBorder(opc);

    bd_fptr border;       border function

void opc_SetBorderFeature(opc, bdfeature);
unsigned int opc_GetBorderFeature(opc);

    unsigned bdfeature;   border features

void opc_SetExplode(opc, explode);
exp_fptr opc_GetExplode(opc);

    exp_fptr explode;     explode function

void opc_SetShadow(opc, shadow);
int opc_GetShadow(opc);

    int shadow;           shadow
```



```

void opc_SetName(opc, name);
char *opc_GetName(opc);

    char *name;          name
void opc_SetMouse(opc, mou);
mouhandler_fptr opc_GetMouse(opc);

    mouhandler_fptr mou;  mouse handler
void opc_SetMouseFeature(opc, moufeature);
unsigned int opc_GetMouseFeature(opc);

    unsigned moufeature   mouse features
void opc_SetSpecial(opc, special);
spc_fptr opc_GetSpecial(opc);

    spc_fptr special;     special function
void opc_SetAux(opc, aux);
aux_fptr opc_GetAux(opc);

    aux_fptr aux;        auxiliary function
void opc_SetKeepwin(opc, keepwin);
boolean opc_GetKeepwin(opc);

    boolean keepwin;     return window handle?
void opc_SetSetwin(opc, setwin);
win_type opc_GetSetwin(opc);

    win_type setwin;     window handle, if returned

```

Of these elements, the only two that differ from standard sed parameters are *keepwin* and *setwin*. *keepwin*, when TRUE, tells the **opc\_** routine to return the popup's handle so you can manipulate it yourself; otherwise, the routine displays and then closes the popup. If you enable *keepwin*, you must display, activate, and close the popup yourself; *setwin* is the handle to the popup that is returned if you enable *keepwin*.

The attribute routines for manipulating the **opc\_struct** refer to either the popup's graphics mode colors or text mode colors. This difference is denoted by the final "G" or "T" in each routine's name:

```

void opc_SetBackAttrG(opc, backg);
byte opc_GetBackAttrG(opc);

    byte backg;          graphics mode background attribute

```

```

void opc_SetRegAttrG(opc, regg);
byte opc_GetRegAttrG(opc);

    byte regg;                graphics mode regular attribute

void opc_SetSelAttrG(opc, selg);
byte opc_GetSelAttrG(opc);

    byte selg;                graphics mode selected attribute

void opc_SetHiRegAttrG(opc, hiregg);
byte opc_GetHiRegAttrG(opc);

    byte hiregg;             graphics mode highlight regular attribute

void opc_SetHiSelAttrG(opc, hiselg);
byte opc_GetHiSelAttrG(opc);

    byte hiselg;             graphics mode highlight selected attribute

void opc_SetBorderAttrG(opc, borderg);
byte opc_GetBorderAttrG(opc);

    byte borderg;            graphics mode border attribute

void opc_SetShadowAttrG(opc, shadowg);
byte opc_GetShadowAttrG(opc);

    byte shadowg;            graphics and text mode shadow attribute

void opc_SetBackAttrT(opc, backt);
byte opc_GetBackAttrT(opc);

    byte backt;              text mode background attribute

void opc_SetRegAttrT(opc, regt);
byte opc_GetRegAttrT(opc);

    byte regt;                text mode regular attribute

void opc_SetSelAttrT(opc, selt);
byte opc_GetSelAttrT(opc);

    byte selt;                text mode selected attribute

void opc_SetHiRegAttrT(opc, hiregt);
byte opc_GetHiRegAttrT(opc);

    byte hiregt;             text mode highlight regular attribute

void opc_SetHiSelAttrT(opc, hiselt);
byte opc_GetHiSelAttrT(opc);

    byte hiselt;             text mode highlight selected attribute

```

```

void opc_SetBorderAttrT(opc, bordert);
byte opc_GetBorderAttrT(opc);

    byte bordert;           text mode border attribute

```

There are also routines that refer to groups of colors, which are:

```

opc_SetColorsG(opc_struct, regular, background, selected);
opc_GetColorsG(opc_struct, regular, background, selected);

    byte regular;           graphics mode regular field attribute
    byte background;       graphics mode text attribute
    byte selected;         graphics mode selected field attribute

opc_SetColorsT(opc_struct, regular, background, selected);
opc_GetColorsT(opc_struct, regular, background, selected);

    byte regular;           text mode regular field attribute
    byte background;       text mode text attribute
    byte selected;         text mode selected field attribute

opc_SetHiColorsG(opc_struct, hireg, hisel);
opc_GetHiColorsG(opc_struct, hireg, hisel);

    byte hireg;            graphics mode regular highlight attribute
    byte hisel;            graphics mode selected highlight attribute

opc_SetHiColorsT(opc_struct, hireg, hisel);
opc_GetHiColorsT(opc_struct, hireg, hisel);

    byte hireg;            text mode regular highlight attribute
    byte hisel;            text mode selected highlight attribute

```

Its position/dimension routines operate on either pixels or character coordinates:

```

void opc_SetPixRow(opc, pixrow);
opcoord opc_GetPixRow(opc);

    opcoord pixrow;        leftmost pixel position

void opc_SetPixCol(opc, pixcol);
opcoord opc_GetPixCol(opc);

    opcoord pixcol;        uppermost pixel position

void opc_SetPixHeight(opc, pixheight);
opcoord opc_GetPixHeight(opc);

    opcoord pixheight;     height in pixels

void opc_SetPixWidth(opc, pixwidth);
opcoord opc_GetPixWidth(opc);

    opcoord width;         width in pixels

```

```

opc_SetRow(opc_struct, row);
opc_GetRow(opc_struct);

    int row;                uppermost position
opc_SetCol(opc_struct, col);
opc_GetCol(opc_struct);

    int col;                leftmost position
opc_SetHeight(opc_struct, height);
opc_GetHeight(opc_struct);

    int height;            height in characters
opc_SetWidth(opc_struct, width);
opc_GetWidth(opc_struct);

    int width;            width in characters

```

Due to the simplicity and large volume of these **opc\_** routines, there are no individual reference pages for each of them. Each function has a **sed\_** analog that you can consult. If you need information on, for instance, **opc\_GetWidth**, consult the reference page for **sed\_GetWidth**.

Finally, remember that you can have multiple **opc\_structs**, if you want different kinds of popups to have various custom “looks.”

## opc\_Close

Destroy an opc structure

---

### Synopsis

```
#include <time.h>
#include "popdecl.h"

void opc_Close(myopc);
opc_type myopc;          the opc structure
```

### Description

This routine destroys an `opc_struct` and frees its memory.

### Return Value

This routine has no return value.

### See Also

**opc\_Open**, **opc\_Text**, **opc\_** routines

### Example

```
opc_type my_opc;

/* ... */

my_opc = opc_Open();

opc_SetColors(my_opc, 0x13, 0x13, 0x31);

/* ... */

opc_Close(my_opc);
```

## Synopsis

```
#include <time.h>
#include "popdecl.h"

opc_Edit(my_opc, title, label, text, tlen);

opc_type my_opc;      Oakland pop context structure
char *title;          title for popup
int label;            help label for popup
char *text;           the text to be edited
unsigned tlen;        maximum length of the buffer
```

## Description

This routine creates and displays a popup editor window. Within the window, it displays word-wrapped text, which the user can edit. To exit the editor, the user clicks the mouse outside of the popup or presses **Esc**.

*text* is a '\0' terminated character array with a '\n' at the end of each line. *tlen* is maximum length of the character array. After calling **opc\_Edit**, *text* contains the edited text. *title* is the title string that the popup's border displays. *label* is the popup's label for the C-scape help system.

To set the editor's position, dimensions, and other parameters, use the various **opc\_Set** routines that are available.

## Return Value

There is no return value.

## See Also

**opc\_View**, **opc\_Text**, **opc\_Open**, **opc\_Close**, **opc\_**

## Example

```
#include <time.h>
#include "popdecl.h"

char text[1025]; /* 1K buffer to hold text */
unsigned int tlen;
char *filename;

fp = fopen(filename, "r");

len = fread(text, 1, 1024, fp);
text[len] = '\0';

fclose(fp);

opc_Edit(NULL, filename, 0, text, len);
```

## Synopsis

```
#include <time.h>
#include "popdecl.h"

boolean opc_FileBox(my_opc, title, label, path, plen, mask);

opc_type my_opc;      Oakland pop context structure
char *title;          title string for the popup
int label;            help label for the popup
char *path;           default path for file listing
int plen;             maximum length of the path
char *mask;           file selection mask
```

## Description

This routine creates and displays a file selection box. *path* describes the default path displayed in the file box, where *plen* is the maximum length of this path. *mask* is the mask that determines which files will appear in the box (such as *\*.txt*, *\*.c*, or *\*.lnf*). The box also includes  and  buttons. *title* is the title string that the popup's border displays. *label* is the popup's label for the C-scape help system.

To set the popup's position, dimensions, and other parameters, use the various **opc\_Set** routines that are available.

## Return Value

Returns TRUE if successful; FALSE, otherwise.

## See Also

**opc\_Open**, **opc\_Close**, **opc\_**

## Example

```
#include <time.h>
#include "popdecl.h"

char filespec[OFFILE_MAXSPEC + 1];

opc_FileBox(NULL, "File Selection Box", 0, filespec, OFFILE_MAX-
SPEC, "*.pcx");
```



### Synopsis

```
#include <time.h>
#include "popdecl.h"

int opc_Menu(my_opc, title, label, choice);

opc_type my_opc;      Oakland pop context structure
char *title;          title for popup
int label;            help label for popup
char **choice;        NULL terminated array of menu choices
```

### Description

This routine creates and displays a popup menu. The mouse or the arrow keys may be used to traverse the menu. To remove the popup without selecting one of the choices, click the mouse outside of the menu or press **Esc**.

*choice* is a NULL terminated list of pointers to the text of the choices. *title* is the title string that the popup's border displays. *label* is the popup's label for the C-scape help system.

To set the editor's position, dimensions, and other parameters, use the various **opc\_Set** routines that are available.

### Return Value

Returns the number of the choice made (first choice is 1) or 0 if **Esc** was pressed.

### See Also

**opc\_View**, **opc\_Text**, **opc\_Open**, **opc\_Close**, **opc\_**

## Example

```
#include <time.h>
#include "popdecl.h"

static char *file_choices[] = {
    "Save file",
    "Load file",
    "Copy file",
    NULL
};

void deal_with_files()
{
    switch (opc_Menu(NULL, "Make a selection", 0, file_choices)) {
        case 1:
            save_file();
            break;
        case 2:
            load_file();
            break;
        case 3:
            copy_file();
            break;
        case 0:
        default:
            break;
    }
}
```

### Synopsis

```
#include <time.h>
#include "popdecl.h"

void opc_Message(my_opc, title, label, msg);

opc_type opc;           Oakland pop context structure
char *title;            title for popup
int label;              help label for popup
char *msg;              the message to be displayed
```

### Description

This high level routine creates and displays the string *msg* in a popup message box. The message is word-wrapped to fit into the window. “\n” characters can be placed into the message string to force line breaks.

*title* is the title string that the popup’s border displays. *label* is the popup’s label for the C-scape help system.

A call to **opc\_Message** places the message box on the display. To remove the message call **opc\_Message** again, either with a new message or with *msg* equal to NULL.

To set the popup’s position, dimensions, and other parameters, use the various **opc\_Set** routines that are available.

### Return Value

There is no return value.

### See Also

**opc\_Text**, **opc\_Verify**, **opc\_Prompt**, **opc\_Open**, **opc\_Close**, **opc\_**

## Example

```
#include <time.h>
#include "popdecl.h"

/* ... */

opc_Message(NULL, NULL, 0, "Loading file:\n Please wait...");

fread(buffer, SIZE, 10000, fp);

opc_Message(NULL, NULL, 0, NULL);

/* ... */
```

### Synopsis

```
#include <time.h>
#include "popdecl.h"

opc_type opc_Open();
```

### Description

This routine creates an `opc_struct`. The `opc_struct` is used by the **opc\_** family of routines. Consult the *C-scape Manual* for more information.

### Return Value

Returns a pointer to the newly created `opc_struct`.

### See Also

**opc\_Close**, **opc\_Edit**, **opc\_Menu**, **opc\_Message**, **opc\_FileBox**, **opc\_Prompt**, **opc\_Text**, **opc\_Verify**, **opc\_View**, **opc\_** routines

## Example

```
opc_type my_opc;

/* ... */

my_opc = opc_Open();

opc_SetColors(my_opc, 0x13, 0x13, 0x31);

if (opc_Verify(my_opc, "Hard disk delete", 0,
    "Okay to delete contents of your hard disk?")) {

    /* they want to delete the hard disk */
    hard_disk_delete();

}
else {

    /* they don't */
    opc_Message(my_opc, "Surprise!", 0,
        "We're going to do it anyway!");

    hard_disk_delete();

}

opc_Close(my_opc);
```

### Synopsis

```
#include <time.h>
#include "popdecl.h"

void opc_Prompt(my_opc, title, label, msg);

opc_type my_opc;      Oakland pop context structure
char *title;          title for popup
int label;            help label for popup
char *msg;            the prompt message to be displayed
```

### Description

This routine creates and displays a popup prompt box. The prompt message is word-wrapped to fit and an **OK** button is placed at the bottom of the window. You can place “\n” characters in the message string to force line breaks. The prompt appears until the user clicks the **OK** button, clicks outside of the popup, or presses **←** or **Esc**. Then the prompt disappears and the function returns.

*title* is the title string that the popup’s border displays. *label* is the popup’s label for the C-scape help system.

To set the popup’s position, dimensions, and other parameters, use the various **opc\_Set** routines that are available.

### Return Value

There is no return value.

### See Also

**opc\_Message**

## Example

```
#include <time.h>
#include "popdecl.h"

char *msg;

if ((fp = fopen(filename, "r")) == NULL) {
    opc_Prompt(NULL, "Warning", 0, "Unable to open file.");
}
```



### Synopsis

```
#include <time.h>
#include "popdecl.h"

sed_type opc_Text(my_opc, title, label, text);

opc_type my_opc;      Oakland pop context structure
char *title;          title for popup
int label;            help label for popup
char *text;           the text to be displayed
```

### Description

This routine creates a popup sed containing a message. The message is word-wrapped to fit into the sed. You can place “\n” characters in the message string to force line breaks.

*title* is the title string that the popup’s border displays. *label* is the popup’s label for the C-scape help system.

**opc\_Text** creates and displays the popup sed and returns a handle to it. You can use any of the other **sed\_** routines to modify the sed as desired. As the sed has no fields in it, calling **sed\_Go** is ill-advised. To remove and destroy the sed, call **sed\_Close**.

You can create as many popup sedes as you like with **opc\_Text**.

To set the popup’s position, dimensions, and other parameters, use the various **opc\_Set** routines that are available.

### Return Value

Returns a handle to a sed or NULL if unable to create a sed.

### See Also

**opc\_Message**, **sed\_Close**

## Example

```
#include <time.h>
#include "popdecl.h"

/* ... */

sed_type msgsed;

msgsed = opc_Text(NULL, NULL, 0, "Please wait");
sed_Repaint(msgsed);

/* ... */

sed_Close(msgsed);
```

### Synopsis

```
#include <time.h>
#include "popdecl.h"

boolean opc_Verify(my_opc, title, label, text);

opc_type my_opc;      Oakland pop context structure
char *title;          title for popup
int label;            help label for popup
char *text;           the text to be viewed
```

### Description

This routine creates and displays a prompt box with the given title along with  and  buttons. To select "OK", the user presses  when control is in that button or clicks on it; clicking outside the box and pressing  are equivalent to selecting the  button.

*title* is the title string that the popup's border displays. *label* is the popup's label for the C-scape help system.

To set the popup's position, dimensions, and other parameters, use the various **opc\_Set** routines that are available.

### Return Value

Returns TRUE if the user selects the  button; FALSE, otherwise.

## Example

```
#include <time.h>
#include "popdecl.h"

static char text[1001];
FILE roland.txt;

/* ... */

if (opc_Verify(NULL, NULL, 0, "Okay to quit?") == TRUE) {
    fwrite(text, sizeof(char), sizeof(text), roland.txt);
}

/* ... */
```

### Synopsis

```
#include <time.h>
#include "popdecl.h"

void opc_View(my_opc, title, label, text);

opc_type my_opc;      Oakland pop context structure
char *title;          title for popup
int label;            help label for popup
char *text;           the text to be viewed
```

### Description

This routine creates and displays a popup box with the given title. The specified text appears within the box and is scrollable in both directions if necessary. To exit the editor the user clicks the mouse outside of the popup or presses **Esc**.

*title* is the title string that the popup's border displays. *label* is the popup's label for the C-scape help system.

To set the popup's position, dimensions, and other parameters, use the various **opc\_Set** routines that are available.

### Return Value

There is no return value.

## Example

```
#include <time.h>
#include "popdecl.h"

static char text[1001];
long len;
opc_type my_opc;
char *filename;

/* ... */

fp = fopen(filename, "r");

len = fread(text, sizeof(char), sizeof(text), fp);
text[len] = '\0';

opc_View(NULL, filename, 0, text);

/* ... */
```

**Synopsis**

```
#include <time.h>

#include "cstime.h"

TIME_T ott_Init(tt);

    TIME_T *tt;           pointer to the TIME_T variable to ini-
                           tialize
```

**Description**

This routine returns the `TIME_T` value that represents the time: midnight, December 31, 1969. If the pointer argument is not `NULL`, the return value is also stored at this address.

**Return Value**

Returns the `TIME_T` value that represents the time midnight, December 31, 1969.

**See Also**

**ott\_Now**, **tm\_Init**, **tm\_Now**

**Note**

The format used to store the value of the `TIME_T` variable is system dependent. If the system's compiler is Standard C compliant, the value is stored using the systems native format. In this case, you are free to mix OWL function calls, C-scape field functions, and calls to your system's libraries (like using **time** to initialize a field function's variable), since all functions use the same format for `TIME_T` variables. On other systems, the value is stored in accordance with the POSIX standard. The library functions on these systems may use a different format to store values in `TIME_T` variables which would make OWL functions and C-scape field functions incompatible with system library functions. By always restricting yourself to only OWL library functions (like using **ott\_Now** in place of **time**), you will maintain the highest level of portability whether your target system is Standard C compliant, or not.

## Example

```
TIME_T tt;  
  
/* initialize time */  
ott_Init(&tt);
```



### Synopsis

```
#include <time.h>

#include "ctime.h"

TIME_T ott_Now(tt);

    TIME_T *tt;           pointer to the TIME_T variable to ini-
                           tialize
```

### Description

This routine returns the `TIME_T` value that represents the current time and date. If the pointer argument is not `NULL`, the return value is also stored at this address.

### Return Value

Returns the `TIME_T` value that represents the current time and date, or the value `-1` cast to a `TIME_T` if not available.

### See Also

**ott\_Init, tm\_Now, tm\_Init**

### Note

The format used to store the value of the `TIME_T` variable is system dependent. If the system's compiler is Standard C compliant, the value is stored using the systems native format. In this case, you are free to mix OWL function calls, C-scape field functions, and calls to your system's libraries (like using **time** to initialize a field function's variable), since all functions use the same format for `TIME_T` variables. On other systems, the value is stored in accordance with the POSIX standard. The library functions on these systems may use a different format to store values in `TIME_T` variables which would make OWL functions and C-scape field functions incompatible with system library functions. By always restricting yourself to only OWL library functions (like using **ott\_Now** in place of **time**), you will maintain the highest level of portability whether your target system is Standard C compliant, or not.

## Example

```
TIME_T tt;
```

```
/* set tt to the current time and date */  
ott_Now(&tt);
```

## pc\_GetMode

Get the current PC video mode

---

### Synopsis

```
#include "pcmode.h"

int pc_GetMode();
```

### Description

This routine returns the current hardware video mode.

This routine is only available under MS DOS.

### Return Value

Return the current video mode.

### See Also

**pc\_IsCGA**

### Example

```
#include "pcmode.h"

switch ( pc_GetMode() ) {      /* MS DOS dependent */

/* ... */

}
```

**Synopsis**

```
#include "pcmode.h"

boolean pc_IsVGA();

boolean pc_IsEGA();

boolean pc_IsCGA();

boolean pc_IsMDA();

boolean pc_IsHerc();

boolean pc_IsCompaq();
```

**Description**

These routines determine whether the video hardware can support the capabilities of a specific video adaptor. The routines are listed below:

<b>pc_IsVGA</b>	Test for Video Graphics Array capabilities.
<b>pc_IsEGA</b>	Test for Enhanced Graphics Adaptor capabilities.
<b>pc_IsCGA</b>	Test for Color Graphics Adaptor capabilities.
<b>pc_IsMDA</b>	Test for Monochrome Display Adaptor capabilities.
<b>pc_IsHerc</b>	Test for Hercules Card capabilities.
<b>pc_IsCompaq</b>	Test for Compaq computer.

These routines determine what *capabilities* the video hardware has, not what type of video adaptor is present. For example, if a system has a VGA then **pc\_IsVGA**, **pc\_IsEGA**, and **pc\_IsCGA** will all return TRUE. This is because a VGA is capable of emulating the features of the other adaptors. To best determine the available video support test for the more advanced types first.

These routines are only available under MS-DOS.

## Return Value

Returns TRUE if the video hardware can support the functionality of the specified display type.

## See Also

**pc\_GetMode**

## Example

```
#include "pcmode.h"

if (pc_IsVGA()) {
    /* ... */
}
else if (pc_IsEGA()) {
    /* ... */
}
else if (pc_IsCGA()) {
    /* ... */
}
else {
    /* ... */
}
```

### Synopsis

```
#include "pcmode.h"

void pc_SetRetrace(mode);
    boolean mode;           the PC retrace mode
```

### Description

This routine sets the retrace mode of the PC device interface. Setting *mode* TRUE instructs the video interface to wait for horizontal retrace signals while writing characters to the display; this eliminates “snow” on CGA monitors. Setting *mode* FALSE instructs the video interface to not wait while writing characters to the display; this speeds up video performance.

When **disp\_Init** initializes the device interface it automatically decides whether or not to wait for retrace signals. **pc\_SetRetrace** should only be used if there is a need to override the default setting (such as when using CGA monitors that do not have snow).

This routine is only available under MS-DOS.

### Return Value

There is no return value.

### Note

This routine is implemented as a macro.

## Example

```
#include "pcmode.h"

#include <time.h>
#include "popdecl.h" /* for opc_Message */

opc_Message(NULL, NULL, 0, "Do you mind a little snow?");

if (tolower(ascii(kb_Read())) == 'n') {
    pc_SetRetrace(FALSE);
}
opc_Message(NULL, NULL, 0, NULL);
```

**Synopsis**

```
boolean sed_Alloc(sed);  
    sed_type sed;           sed for which to allocate
```

**Description**

This routine allocates storage space for each of the sed's field variables. It returns TRUE if it was able to allocate the variable space; FALSE if not.

If space was allocated for this sed by a prior call to **sed\_Alloc** or by a call to **sfile\_LoadSed** with the SED\_ALLOC flag set then that variable space is freed before the current allocation. The amount of space allocated for each field is determined by the fields' *varsize* element.

If you wish to start a field with a particular value you must subsequently initialize the field's variable by copying data in the field's storage space. You can get a pointer to this space with **sed\_GetVar** or **sed\_GetNameVar**.

The storage allocated by **sed\_Alloc** is released when you close the sed with **sed\_Close**.

**Return Value**

Returns TRUE if it was able to allocate the variable space; FALSE if the memory was not available for the allocation.

**Note**

This routine is implemented as a macro.

**See Also**

**sfile\_LoadSed**, **sed\_GetVar**, **sed\_GetNameVar**



## Example

```
sed = sed_Open(menu);  
sed_Alloc(sed);  
  
/* set value of field 0 (a string, note cast) */  
strcpy((char *) sed_GetVar(sed, 0), "Starting Value");  
  
/* set value of field 1 (an integer, note cast) */  
*( (int *) sed_GetVar(sed, 1) ) = 22;
```

### **Synopsis**

```
boolean sed_BorderExists(sed);  
    sed_type sed;           the sed
```

### **Description**

This routine determines if there is a border attached to the sed.

### **Return Value**

Returns TRUE if the sed has a border attached to it and FALSE otherwise.

### **See Also**

**sed\_SetBorder**

### **Note**

This routine is implemented as a macro.

### **Example**

```
if (sed_BorderExists(sed) ) {  
    sed_BorderPrompt(sed, "Border already exists!");  
}
```

## sed\_BorderPrompt

Display a prompt in the border

---

### Synopsis

```
void sed_BorderPrompt(sed, string);  
    sed_type sed;           the sed  
    char *string;          the prompt string
```

### Description

This routine tells the border to display the prompt string in its prompt area. The routine does nothing if there is no border attached to the sed or if the sed's border has no prompt area.

### Return Value

There is no return value.

### See Also

**sed\_SetBorder, sed\_SetBorderFeature**

### Note

This routine is implemented as a macro.

### Example

```
sed_BorderPrompt(sed, "Invalid Entry!");  
  
while(!kb_Check()) {  
    ;  
}  
  
sed_BorderPrompt(sed, NULL); /* Clear the border prompt */
```

### Synopsis

```
void sed_Center(sed);  
    sed_type sed;           the sed
```

### Description

This routine repositions a sed so that it is centered on the display.

### Return Value

There is no return value.

### See Also

[sed\\_SetPosition](#)

### Example

```
sed = sed_Open(menu);  
  
sed_Center(sed);  
sed_Repaint(sed);  
sed_Go(sed);
```

### Synopsis

```
void sed_ClearTB(sed);  
    sed_type sed;           the sed
```

### Description

This routine destroys the sed's text buffer and replaces it with a new, empty one with default settings.

This routine is usually used before using **sed\_SetTB** to insert new text into a text buffer.

### Return Value

This routine has no return value.

### Note

This routine is implemented as a macro.

### See Also

**sed\_GetTB, sed\_SetTB**

### Example

```
char buffer[BUFLen];  
  
/* Clear contents of the text buffer */  
sed_ClearTB(sed);  
  
/* Copy buffer into the text buffer */  
sed_SetTB(sed, buffer, BUFLen);  
  
/* ... */
```

### Synopsis

```
void sed_Close(sed);  
    sed_type sed;           the sed object
```

### Description

This routine closes the sed and releases any storage it used. The sed can not be used after it has been closed.

If the sed's window is employed, **sed\_Close** fires it and removes the sed's image from the display.

**sed\_Close** releases all the storage used by the objects attached to the sed. The menu from which the sed was created is closed and can no longer be used.

If there are bobs attached to the sed, **sed\_Close** will close them. If you used **sled\_Open** to open the sed, **sed\_Close** will close all the sled's column arrays.

### Return Value

There is no return value.

### See Also

**sed\_Open**

### Note

This routine is implemented as a macro.

## Example

```
void get_name(name)
    char *name;
/*
    Ask for and get a name from the user.
*/
{
    menu_type menu;
    sed_type sed;

    /* create the menu */
    menu = menu_Open();
    menu_Printf(menu, "Enter your name: @f[#####]",
        name, &string_funcs);

    menu_Flush(menu);

    /* create the sed */
    sed = sed_Open(menu);

    /* display the sed */
    sed_Repaint(sed);

    /* activate the sed; get the user input */
    sed_Go(sed);

    /* release storage used by the sed and menu objects */
    sed_Close(sed);
}
```

### Synopsis

```
bob_type sed_CreateBob(sed, mode);  
  
    sed_type sed;           the sed  
    int mode;              the dependence flag
```

### Description

This routine creates a bob object from a sed.

Bobs (*basic objects*) provide a standard interface to a number of different objects. Bobs are created from objects, such as sedes, that you can place onto the display and activate.

The *mode* argument can have one of the following values:

- |                 |  |
|-----------------|--|
| BOB_DEPENDENT   | creates a bob that depends upon the sed to which it is attached. A dependent bob is repainted when its owner is repainted and moves when its owner moves. You can create embedded screens, such as text editors (teds) and scrolling lists (sleds), with dependent bobs. |
| BOB_INDEPENDENT | creates a bob that is independent of its owner. The framer and slug menuing systems use independent bobs.  |

Bobs allow you to create complex screens consisting of various screen types working in concert. Any C-scape field can contain a handle to a bob object. **menu\_Printf** or **sed\_SetFieldBob** can both set this handle. When a sed is repainted all the dependent bobs connected to it are repainted as well. A field function can pass control to a bob by calling **bob\_Go**. Refer to the manual for more information on bob objects.

Bobs can create screens with note pad editors, scrollable sub-regions, or other sub regions.

Once you have created a bob from a sed, you can use the **bob\_** functions to control it.



## Return Value

Returns the bob object created from the sed or NULL if the bob object cannot be created.

## Note

This routine is implemented as a macro.

## See Also

**bob\_GetSed, bob\_Go, menu\_Printf, sed\_GetFieldBob**

## Example

```
sed_type sed;
bob_type bob;

/* ... */

sed = sed_Open(menu);

menu_Printf(outer_menu, "@fb[]", NULL, &bob_funcs,
    sed_CreateBob(sed, BOB_DEPENDENT));
```

**Synopsis**

```
boolean sed_DecChar(sed);  
sed_type sed;           the sed object
```

**Description**

This routine moves the cursor to the previous character in a field. If the cursor is already at the first position in the field, the routine does nothing.

**Return Value**

This routine returns TRUE if it succeeds in moving the cursor, otherwise it returns FALSE.

**See Also**

**sed\_GetRecordPos, sed\_GetMergePos, sed\_GoEnd, sed\_GoHome, sed\_GoToChar, sed\_IncChar**

**Example**

```
void string_fkey(sed)  
{  
    sed_type sed;  
    int scancode;  
  
    switch (scancode = kb_Read()) {  
    case LEFT:  
        /* move the cursor backwards if the left arrow is pressed */  
        sed_DecChar(sed);  
        break;  
    case RIGHT:  
        /* move the cursor forwards if the right arrow is pressed */  
        sed_IncChar(sed);  
        break;  
  
        /* .... */  
    }  
}
```

### Synopsis

```
int sed_DecField(sed);  
    sed_type sed;           the sed
```

### Description

This routine takes the field before the current field and makes it the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_DecField** does the following. If the sed is in active mode, it calls the **fexit** function for the current field. If **fexit** returns FALSE, it does nothing and returns SED\_INVALID. If the current field is the first unprotected field the routine does nothing and returns SED\_STUCK. Otherwise, the preceding field becomes the new current field. Before it enters the new field, **sed\_DecField** calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_DecField** does not call the **fexit** and **fenter** functions.

You must not call this routine in either a field's **fenter** or **fexit** routines.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. A “p” following the number represents a protected field. Calling **sed\_DecField** from field 4 would make field 3 the new current field. Calling **sed\_DecField** from field 3 would make field 1 the new current field (field 2 is protected).

0	1	2p
3	4	5
6	7	8

## Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it could not find an unprotected new field to which it can move.

## See Also

**sed\_GetFieldNo, sed\_GotoField, sed\_IncField, sed\_ProtectField**

## Example

```
boolean inter_field(sed, scancode)
    sed_type sed;
    int scancode;

{
    switch (scancode) {
        /* ... */
        case UP:
            sed_DecField(sed);
            return(TRUE);
        case DOWN:
            sed_IncField(sed);
            return(TRUE);
        default:
            break;
    }
    return(FALSE);
}
```

### Synopsis

```
void sed_DeleteField(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;            the field number
```

### Description

This routine removes a field from the sed's menu. All the field numbers after *fieldno* are decremented by one. If the current field is deleted then the next field becomes the current field.

### Return Value

There is no return value.

### See Also

**sed\_DeleteRows**

### Example

```
switch ( kb_Read() ) {  
  
case DEL:  
    sed_DeleteField(sed, sed_GetFieldNo(sed));  
    sed_Repaint(sed);  
    break;  
  
/* ... */
```

### Synopsis

```
void sed_DeleteRows(sed, row, count);  
    sed_type sed;           the sed  
    int row;                starting row to delete  
    int count;              number of rows to delete
```

### Description

This routine deletes *count* rows from the sed starting at menu row *row*. All the fields and text in the rows are deleted. The field numbers of fields after the deleted fields are decreased accordingly.

### Return Value

There is no return value.

### See Also

`sed_DeleteField`, `sed_InsertRows`

### Example

```
switch( kb_Read() ) {  
  
case DEL:  
    sed_DeleteRows(sed,  
        menu_GetFieldRow(sed, sed_GetFieldNo(sed)), 1);  
    sed_Repaint(sed);  
    break;  
  
/* ... */
```

### Synopsis

```
int sed_DoAux(sed, msg, indata, outdata);

    sed_type sed;           the sed
    int msg;               the auxiliary message
    void *indata;          pointer to incoming data
    void *outdata;         pointer to outgoing data
```

### Description

This routine calls the sed's auxiliary function. *msg* is an integer specifying an action for the auxiliary function to perform. *indata* is a pointer to incoming data to be used by the auxiliary function. *outdata* is a pointer to outgoing data returned by the auxiliary function. **sed\_DoAux** does nothing if the sed has no auxiliary function.

The auxiliary function is intended to provide the programmer with additional control over the operations of a sed. C-scape sends the following messages to auxiliary functions automatically:

- |                |   |
|----------------|---|
| SED_PRESENTER  | called by a <b>_Repaint</b> function immediately before the sed's <b>sender</b> functions are called. The auxiliary function's return value is ignored when it is sent this message.  |
| SED_POSTSENDER | called by a <b>_Repaint</b> function immediately after the sed's <b>sender</b> functions are called. The auxiliary function's return value is ignored when it is sent this message.   |
| SED_PRESEXIT   | called by <b>sed_Go</b> immediately before the sed's <b>sexit</b> functions are called. You can use this message to provide validation for an entire sed. If the auxiliary function returns <b>FALSE</b> in response to this message, the user will not be able to leave the sed. |
| SED_POSTSEXIT  | called by <b>sed_Go</b> immediately after the sed's <b>sexit</b> functions are called. The auxiliary function's return value is ignored when it is sent this message.   |

SED_PREFENTER	called immediately before a field's <b>fenter</b> function is called. The auxiliary function's return value is ignored when it is sent this message.
SED_POSTFENTER	called immediately after a field's <b>fenter</b> function is called. The auxiliary function's return value is ignored when it is sent this message.
SED_PREFEXIT	called immediately before a field's <b>fexit</b> function is called. This message can provide extra field validation. If the auxiliary function returns 0 when it receives this message, it is functionally equivalent to the <b>fexit</b> function returning FALSE.
SED_POSTFEXIT	called immediately after a field's <b>fexit</b> function is called (if the <b>fexit</b> function was successful). This message can provide extra field validation; if the auxiliary function returns FALSE when it receives this message, it is functionally equivalent to the <b>fexit</b> function returning FALSE.

These standard C-scape messages do not use *indata* or *outdata* (they are NULL).

C-scape provides one standard auxiliary function, called **aux\_Top**. It raises its sed above all other windows on the display when it becomes current by calling **sed\_Top** upon receiving a WINA\_STARTGO message.

You can use **sed\_DoAux** to send your own custom messages to an auxiliary function. Your message values should be unique and greater than SED\_LASTMSG.

Note that there are also auxiliary messages at the window and object levels. These are documented in the *OWL Manual* and *OWL Function Reference*.

## Return Value

Returns the value returned by the auxiliary function upon its completion.

## Note

This routine is implemented as a macro.



## See Also

### sed\_SetAux

## Example

```
#define MYSED_RESETFIELDS      SED_LASTMSG + 1

int aux_Reset(sed, msg, indata, outdata)
    sed_type sed;
    int msg;
    VOID *indata;
    VOID *outdata;
/*
    This auxiliary function resets the sed's fields when it receives
    a MYSED_RESETFIELDS message.
*/
{
    int fldno;

    switch(msg) {
    case MYSED_RESETFIELDS:
        /* reset all the field records */
        for (fldno = 0; fldno < sed_GetFieldCount(sed); fldno++) {
            sed_SetRecord(sed, "", fldno);
        }
        sed_UpdateFields(sed);
        break;
    }

    return(1);
}

/* ... */

sed_SetAux(sed, aux_Reset);

/* Clear our sed by sending a
   MYSED_RESETFIELDS message to its auxiliary function.
*/

sed_DoAux(sed, MYSED_RESETFIELDS, NULL, NULL);

/* ... */
```

## sed\_DoFieldFenter - sed\_DoFieldSexit

Call a field function

---

### Synopsis

```
void sed_DoFieldFenter(sed, fieldno);
    sed_type sed;          the sed
    int fieldno;           the field number

boolean sed_DoFieldFexit(sed, fieldno);
    sed_type sed;          the sed
    int fieldno;           the field number

void sed_DoFieldFkey(sed, fieldno);
    sed_type sed;          the sed
    int fieldno;           the field number

void sed_DoFieldSenter(sed, fieldno);
    sed_type sed;          the sed
    int fieldno;           the field number

void sed_DoFieldSexit(sed, fieldno);
    sed_type sed;          the sed
    int fieldno;           the field number
```

### Description

- sed\_DoFieldFenter** This routine calls the **fenter** function for the specified field. This causes the SED\_PREFENTER and SED\_POSTFENTER messages to be sent as well.
- sed\_DoFieldFexit** This routine calls the **fexit** function for the specified field. This causes the SED\_PREFEXIT and SED\_POSTFEXIT messages to be sent as well.
- sed\_DoFieldFkey** This routine calls the **fkey** function for the specified field.
- sed\_DoFieldSenter** This routine calls the **senter** function for the specified field. This causes the SED\_PRESENTER and SED\_POSTSENDER messages to be sent as well.

**sed\_DoFieldSexit** This routine calls the **sexit** function for the specified field. This causes the SED\_PRESEXIT and SED\_POSTSEXIT messages to be sent as well.

### **Return Value**

These routines have no return value with the exception of **sed\_DoFieldFexit**, which returns TRUE only if all validation on the field succeeds. (This validation is performed by the **fexit** routine and the auxiliary function.)

### Synopsis

```
void sed_DoSentries(sed);  
    sed_type sed;          the sed
```

### Description

This routine calls the **senter** function for all the fields in a sed.

### Return Value

There is no return value.

### See Also

**sed\_DoSexits**

### Note

This routine is implemented as a macro.

### Example

```
sed_DoSentries(sed);
```

### Synopsis

```
void sed_DoSexits(sed);  
    sed_type sed;           the sed
```

### Description

This routine calls the **sexit** function for all the fields in a sed.

### Return Value

There is no return value.

### See Also

**sed\_DoSeters**

### Example

```
sed_DoSexits(sed);
```

### Synopsis

```
boolean sed_DoSpecial(sed, scancode);  
    sed_type sed;           the sed  
    int scancode;          keystroke for the special function to process
```

### Description

This routine executes the special function attached to the sed. If the sed has no special function the routine returns FALSE.

The special function is used to customize the operation of all the fields in a sed without having to modify any field functions. The **fkey** function of the standard field functions call **sed\_DoSpecial**.

### Return Value

Returns the value returned by the sed's special function. The special function returns TRUE if it intercepted the keystroke, FALSE otherwise.

### See Also

#### sed\_SetSpecial

### Example

```
void string_fkey(sed)  
{  
    sed_type sed;  
    int scancode;  
  
    if ( sed_DoSpecial(sed, scancode) )  
        return;  
    if ( special_key(scancode) )  
        return;  
  
    /* ... */  
}
```

### Synopsis

```
int sed_DownField(sed);  
    sed_type sed;           the sed
```

### Description

This routine takes the field below the current field and makes it the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_DownField** does the following. If the sed is in active mode it calls the **fexit** function for the current field. If **fexit** returns FALSE, it does nothing and returns SED\_INVALID. If the current field is the bottom-most unprotected field the routine does nothing and returns SED\_STUCK. Otherwise, the field below the current field becomes the new current field. Before **sed\_DownField** enters the new field, it calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_DownField** does not call the **fexit** and **fenter** functions.

You must not call this routine in either a field's **fenter** or **fexit** routines.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. A “p” following the number represents a protected field. Calling **sed\_DownField** from field 4 would make field 7 the new current field. Calling **sed\_DownField** from field 2 would make field 8 the new current field (field 5 is protected).

0	1	2
3	4	5p
6	7	8

## Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it could not find an unprotected new field to which it can move.

## See Also

**sed\_UpField, sed\_LeftField, sed\_RightField, sed\_ProtectField**

## Note

This routine is implemented as a macro.

## Example

```
switch (scancode) {
/* ... */
    case UP:
        sed_UpField(sed);
        return(TRUE);
    case DOWN:
        sed_DownField(sed);
        return(TRUE);
    case LEFT:
        sed_LeftField(sed);
        return(TRUE);
    case RIGHT:
        sed_RightField(sed);
        return(TRUE);
/* ... */
}
```



### Synopsis

```
int sed_FindField(sed, boxp, direction);  
  
    sed_type      sed;           sed window  
    obox          *boxp;         character box  
    int           direction;      search direction
```

### Description

This routine searches for a field in *sed*'s menu that lies within the area specified by *boxp*. It is most commonly used within a mouse handler to find the field over which the mouse cursor lies.

*boxp* is a pointer to an obox (*O*akland *c*haracter *b*ox). The coordinates of the box are relative to the *sed*'s menu. The box must take into account the *sed* offsets (*sed\_GetXoffset*, *sed\_GetYoffset*) into the menu.

*direction* indicates a preference for the manner in which the box is searched. It may have one of 4 values with the following significance:

OAK_DOWN or OAK_RIGHT	searches down and right from the top row of the box.
OAK_UP	searches up and right from the bottom row of the box.
OAK_LEFT	searches left and up from the bottom row of the box.

**sed\_FindField** finds the first unprotected field in the given direction. The field may lie fully or partially in the box.

### Return Value

Returns an integer denoting the number of the field found; -1, if no field was found; -2, if the only field found was a protected field.

### Note

An ocbox uses a C-scape data structure with 4 integer members: *toprow*, *botrow*, *leftcol*, and *rightcol*. It describes a rectangular area in character coordinate units. See the *OWL Manual* or the *OWL Function Reference* for more information about ocboxes.

## sed\_GetActive

Checks if the sed currently has control

---

### Synopsis

```
boolean sed_GetActive(sed);  
sed_type sed;           the sed
```

### Description

A sed can become active by any of the following means:

- calling **sed\_Go** on it
- passing control to it with the mouse
- passing control to it with **sed\_SetNextWin**

### Return Value

Returns TRUE if the sed is active; FALSE, otherwise.

### Example

```
if sed_GetActive(sed) {  
  
    /* put it on top if it's active */  
    sed_Top(sed);  
  
} else {  
  
    /* remove it otherwise */  
    sed_Pop(sed);  
}
```

### Synopsis

```
sed_type sed_GetAncestor(sed);  
    sed_type sed;           the sed
```

### Description

This routine gets the most distant ancestor (ultimate owner) of the given sed. If the sed is an orphan (unowned) it gets the given sed itself.

### Return Value

Returns a pointer to a sed. This return value may be the given sed itself if it has no owner.

### Note

This routine is implemented as a macro.

### See Also

**bob\_GetOwner**

### Example

```
sed_type sed, granpa;  
  
/* ... */  
  
granpa = sed_GetAncestor(sed);  
  
if (granpa != sed) {  
    /* owner is our ancestor ... */  
}
```

**Synopsis**

```
unsigned int sed_GetAttrTB(sed, buf, len, mode, attr);  
sed_type sed;           the sed  
char *buf;              the buffer for storing text  
unsigned int len;        the size of the requested buffer  
int mode;               ted line ending save mode  
byte *attr;             address to save the gotten attribute
```

**Description**

This routine retrieves a string of characters from the sed's text buffer, all of which have the same attribute. It stores the string in *buf* and the string's attribute in *attr*.

The function begins retrieving text from the current cursor position. It returns the number of characters retrieved, a value between 0 and *len*. If number of characters from the current cursor position to the end of the text buffer is less than *len*, then the function will return this value.

The *mode* can either be TED\_HARD or TED\_SOFT. If the mode is TED\_HARD, **sed\_GetAttrTB** adds newlines at the end of any word-wrapped lines. If the mode is TED\_SOFT, the function saves only those newlines that are already part of the buffer.

**Return Value**

Returns the number of characters retrieved.

**Note**

This routine is implemented as a macro.

**See Also**

**sed\_GetTB, sed\_SetTB, ted\_WriteFile, sed\_RewindTB**

## Example

```
while ((len = sed_GetAttrTB(sed, buf, buflen, TED_SOFT, &attr)) >
0) {
    printf("attr = %x, length = %d\n", attr, len);
    printf(buf);
    printf("\n");
}
```

### Synopsis

```
aux_fptr sed_GetAux(sed);  
    sed_type sed;    the sed
```

### Description

This routine gets a pointer to the given sed's auxiliary function. Use **sed\_SetAux** to set a sed's auxiliary function pointer.

### Return Value

Returns a pointer to the sed's auxiliary function; FNULL, if none.

### Note

This routine is implemented as a macro.

### See Also

**sed\_SetAux**

### Example

```
if (sed_GetAux(sed) == FNULL) {  
    sed_SetAux(sed, aux_FileManager);  
}
```

### Synopsis

```
int sed_GetBaton(sed);  
    sed_type sed;           the sed
```

### Description

This routine gets the value of the baton. The baton passes state information between the functions in the field function structure (the **fenter**, **fexit**, **fkey**, **senter**, and **sexit** functions) and is as a return value by **sed\_Go**. It is initialized to -1 when the sed is created.

A typical use is in a menu that returns 0 for the **[Esc]** key, 1 for the first menu choice, 2 for the second menu choice, and so on. Another use might be to tell the **fexit** function not to validate the data in a field if the user pressed the **[Esc]** key.

### Return Value

This routine returns the integer value of the baton.

### See Also

**sed\_SetBaton**, **sed\_Go**

### Note

This routine is implemented as a macro.

### Example

```
void my_string_sexit(sed, fieldno)  
    sed_type sed;  
    int fieldno;  
/*  
    Copy the record string back into the native string.  
    Skip if (baton == SED_ABORT).  
*/  
{  
    if (sed_GetBaton(sed) != SED_ABORT) {  
        strcpy(sed_GetVar(sed, fieldno), sed_GetRecord(sed, fieldno));  
    }  
}
```



**Synopsis**

```
bob_type sed_GetBob(sed);  
    sed_type sed;           the sed
```

**Description**

This routine returns a pointer to the bob created from the sed. If no bob was created from the sed it returns NULL.

**Return Value**

Returns a pointer to the bob created from the sed. Returns NULL if no bob has been created from the sed.

**Note**

This routine is implemented as a macro.

**See Also**

**bob\_GetOwner, sed\_CreateBob, sed\_GetAncestor**

**Example**

```
bob_type bob;  
sed_type sed;  
  
/* ... */  
  
bob = sed_GetBob(sed)
```

### Synopsis

```
void sed_GetBordCorners(sed, ul_row, ul_col, lr_row, lr_col);

    sed_type sed;           the sed
    int *ul_row;            the upper left hand row of the sed
    int *ul_col;            the upper left hand column of the sed
    int *lr_row;            the lower right hand row of the sed
    int *lr_col;            the lower right hand column of the sed
```

### Description

This routine determines the upper left and lower right hand corners of the sed's border and places them into the locations pointed to by *ul\_row*, *ul\_col*, *lr\_row*, and *lr\_col*.

The corners are given in display coordinates relative to the upper left hand corner of the display.

The size of the border attached to the sed determines the location of the corners. If there is no border attached to the sed then the size of the sed is used.

### Return Value

There is no return value.

### See Also

**sed\_GetCorners, sed\_GetBorderHeight, sed\_GetBorderWidth**

### Example

```
int ymin, xmin, ymax, xmax;

sed_GetBordCorners(sed, &ymin, &xmin, &ymax, &xmax);
```

### Synopsis

```
byte sed_GetBorderColor(sed);  
    sed_type sed;          the sed
```

### Description

This routine returns the color of the border attached to the sed.

### Return Value

If there is a border attached to the sed its color is returned. If there is no border attached to the sed, 0 is returned.

### See Also

**sed\_SetBorder, sed\_SetBorderColor, sed\_BorderExists**

### Note

This routine is implemented as a macro.

### Example

```
byte b_color;  
  
b_color = sed_GetBorderColor(sed);
```

## **sed\_GetBorderHeight**

Get the height of the sed's border

---

### **Synopsis**

```
int sed_GetBorderHeight(sed);  
    sed_type sed;          the sed
```

### **Description**

This routine returns the height of the sed's attached border. If there is no attached border or if the sed is taller than the border, the height of the sed is returned.

### **Return Value**

The height of the sed's border.

### **See Also**

**sed\_GetHeight, sed\_GetMenuHeight, sed\_GetBorderWidth**

### **Example**

```
int hgt;  
  
hgt = sed_GetBorderHeight(sed);
```

## **sed\_GetBorderWidth**

Get the width of the sed's border

---

### **Synopsis**

```
int sed_GetBorderWidth(sed);  
    sed_type sed;           the sed
```

### **Description**

This routine returns the width of the sed's attached border. If there is no attached border or if the sed is wider than the border, the width of the sed is returned.

### **Return Value**

The width of the sed's border.

### **See Also**

**sed\_GetWidth, sed\_GetMenuWidth, sed\_GetBorderHeight**

### **Example**

```
int wid;  
  
wid = sed_GetBorderWidth(sed);
```

**Synopsis**

```
char sed_GetChar(sed, pos);  
    sed_type sed;           the sed  
    int pos;                the position
```

**Description**

This routine returns the character at record position number *pos* within the current field's record. The first character is at position 0.

**Return Value**

The character at the specified position within the current field's record.

**See Also**

**sed\_GetCurrChar, sed\_GetFieldChar**

**Note**

This routine is implemented as a macro.

**Example**

```
/* Toggle minus sign in field ... */  
  
if (sed_GetChar(sed, 0) == '-') {  
    /* erase minus sign */  
  
    /* ... */  
}
```

### Synopsis

```
void sed_GetColors(sed, regular, background, selected);  
    sed_type sed;           the sed  
    byte *regular;          regular field attribute  
    byte *background;       background attribute  
    byte *selected;         selected field attribute
```

### Description

This routine determines the colors of the sed and places them into the locations pointed to by *regular*, *background*, and *selected*.

*regular*, *background*, and *selected* must be **pointers** to bytes.

### Return Value

There is no return value.

### See Also

**sed\_SetColors**

### Example

```
byte reg, back, sel;  
  
sed_GetColors(sed, &reg, &back, &sel);
```

**Synopsis**

```
void sed_GetCorners(sed, ul_row, ul_col, lr_row, lr_col);  
    sed_type sed;           the sed  
    int *ul_row;            the upper left hand row of the sed  
    int *ul_col;            the upper left hand column of the sed  
    int *lr_row;            the lower right hand row of the sed  
    int *lr_col;            the lower right hand column of the sed
```

**Description**

This routine determines the upper left and lower right hand corners of the sed rectangle and places them into the locations pointed to by *ul\_row*, *ul\_col*, *lr\_row*, and *lr\_col*.

The corners are given in display coordinates relative to the upper left hand corner of the display.

The size of the sed determines the location of the corners. If there is a border attached to the sed, it is ignored.

**Return Value**

There is no return value.

**Example**

```
int ymin, xmin, ymax, xmax;  
  
sed_GetCorners(sed, &ymin, &xmin, &ymax, &xmax);
```



## **sed\_GetCurrChar**

Get the character at the current position

---

### **Synopsis**

```
char sed_GetCurrChar(sed);  
    sed_type sed;          the sed
```

### **Description**

This routine returns the character at the current cursor position within the current field's record.

### **Return Value**

The character at the current position within the current field's record.

### **See Also**

**sed\_GetChar, sed\_GetFieldChar**

### **Note**

This routine is implemented as a macro.

### **Example**

```
if (sed_GetCurrChar(sed) == ' ') {  
    /* ... */  
}
```

### Synopsis

```
VOID *sed_GetFieldData(sed, datano);  
    sed_type sed;           the sed  
    int datano;             the data number
```

### Description

This routine returns a data pointer for the sed's current field. Each field has at least one generic data pointer that the programmer can set. The data pointer is a VOID \* pointer. You can use the *datano* argument to indicate which data pointer you want (0 is the first data pointer).

A field normally has one data pointer but you can override this with the **menu\_Printf** “@fd” command when the field is defined.

The standard field functions use the field generic data pointers in the following ways:

- (0) to store border prompt strings.
- (1) to store validation data.
- (2) to store formatting information.

### Return Value

Returns the generic data pointer for the field. The generic data pointer is a VOID \* pointer. Use a type cast to convert it to a different type if necessary.

Returns NULL if *datano* exceeds the number of data pointers associated with the field.

### See Also

**menu\_Printf**, **sed\_SetFieldData**, **sed\_GetFieldDataCount**, **sed\_GetFieldData**

### Note

This routine is implemented as a macro.

## Example

```
boolean int_fexit(sed)
    sed_type sed;
/*
    Validates an integer using the string in field data 1.
*/
{
    int val;

    sscanf(sed_GetCurrRecord(sed), "%d", &val);
    if ( !valid_Int(val,
        (char *) sed_GetFieldData(sed, sed_GetFieldNo(sed), 1)) ) {

        tone();
        sed_BorderPrompt(sed, "Number out of range");
        /* wait for a keystroke */
        while (!kb_Check())
            ;
        sed_BorderPrompt(sed,
            sed_GetFieldData(sed, sed_GetFieldNo(sed), 0));

        return(FALSE);
    }

    sed_DoFieldSexit(sed, sed_GetFieldNo(sed));
    sed_DoFieldSenter(sed, sed_GetFieldNo(sed));
    sed_UpdateCurrField(sed);
    sed_BorderPrompt(sed, NULL);
    return(TRUE);
}
```

### Synopsis

```
char *sed_GetCurrMerge(sed);  
    sed_type sed;          the sed
```

### Description

This routine returns the merge of the current field. A merge is the string containing the entire contents of the field, both the writeable and nonwriteable positions.

The merge should be copied into a buffer before being used; the contents of the actual merge should not be altered.

### Return Value

Returns a pointer to the merge string of the current field.

### See Also

**sed\_GetMerge, sed\_GetRecord, sed\_GetScratchPad**

### Note

This routine is implemented as a macro.

### Example

```
char *p;  
  
p = sed_GetScratchPad(sed);  
  
strcpy(p, sed_GetCurrMerge(sed));
```

**Synopsis**

```
char *sed_GetCurrRecord(sed);  
    sed_type sed;           the sed
```

**Description**

This routine returns the record of the current field. A record consists of only the writeable positions in the field.

**Return Value**

Returns a pointer to the record string of the current field.

**See Also**

**sed\_SetCurrRecord, sed\_GetRecord, sed\_GetMerge, sed\_GetScratchPad**

**Note**

This routine is implemented as a macro.

**Example**

```
char buffer[101];    /* make sure buffer is long enough */  
  
strcpy(buffer, sed_GetCurrRecord(sed));
```

**Synopsis**

```
int sed_GetCurrRecordLen(sed);  
    sed_type sed;          the sed
```

**Description**

This routine returns the length of the record string of the current field. A record consists of only the writeable positions in the field.

**Return Value**

Returns the length of the current field's record string.

**See Also**

**sed\_GetRecord, sed\_GetRecordLen**

**Note**

This routine is implemented as a macro.

**Example**

```
int len;  
  
len = sed_GetCurrRecordLen(sed);
```

**Synopsis**

```
VOID *sed_GetCurrVar(sed);  
    sed_type sed;           the sed
```

**Description**

This routine returns the pointer to the variable for the current field. The pointer is bound to the field during the calls to **menu\_Printf**.

**Return Value**

This routine returns a pointer to the variable for the current field. The field's variable is a VOID \* pointer. Use a type cast to convert it to another type when necessary.

**See Also**

**sed\_GetVar**

**Note**

This routine is implemented as a macro.

**Example**

```
void int_senter(sed, fieldno)  
    sed_type sed;  
    int fieldno;  
/*  
  Convert native type to string for record.  
*/  
{  
    char *s;  
  
    s = sed_GetScratchPad(sed);  
  
    sprintf(s, "%d", *((int *) sed_GetVar(sed, fieldno)));  
  
    strright(s, sed_GetRecordLen(sed, fieldno));  
    sed_SetRecord(sed, s, fieldno);  
}
```

### Synopsis

```
unsigned int sed_GetCursorType(sed);  
    sed_type sed;           the sed
```

### Description

This routine determines the current size of the sed's cursor. The sed's cursor type is initialized to `CURSOR_NORMAL`.

### Return Value

Returns the sed's current cursor type.

The cursor type is one of the following values:

<code>CURSOR_NORMAL</code>	The standard cursor.
<code>CURSOR_NONE</code>	An invisible cursor.
<code>CURSOR_BLOCK</code>	A full-sized cursor.
<code>CURSOR_DASH</code>	A thin cursor located in the middle of the character location.
<code>CURSOR_HALF</code>	A cursor filling half of the character location.
<code>CURSOR_THIN</code>	A thin cursor.

### See Also

`sed_SetCursorType`

### Example

```
unsigned int ctype;  
  
ctype = sed_GetCursorType(sed);
```



**Synopsis**

```
VOID *sed_GetData(sed);  
    sed_type sed;           the sed
```

**Description**

This routine returns the sed's generic data pointer. The generic data pointer is a VOID \* pointer that you can use for attaching program-specific data to a sed.

Some typical uses for the generic data pointer are:

- (1) as a place to store an array of strings to be displayed as messages, and
- (2) as a place to store the last value of the field so that it can be recalled in case the user makes a mistake.

**Return Value**

Returns the generic data pointer for the sed. The generic data pointer is a VOID \* pointer. Use a type cast to convert it to a different type if necessary.

**See Also**

**sed\_SetData**

**Note**

This routine is implemented as a macro.

**Example**

```
struct my_struct *my_s;  
  
my_s = (struct my_struct *) sed_GetData(sed);
```

### Synopsis

```
boolean sed_GetExit(sed);  
    sed_type sed;          the sed object
```

### Description

This routine returns the exit state of the sed.

### Return Value

The exit state of the sed.

### See Also

**sed\_SetExit, sed\_ToggleExit**

### Note

This routine is implemented as a macro.

### Example

```
boolean exit;  
  
exit = sed_GetExit(sed);
```

### Synopsis

```
bob_type sed_GetFieldBob(sed, fieldno);  
  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### Description

This routine gets a handle to the bob object attached to field *fieldno*.

### Return Value

Returns a handle to the bob object attached to the field. Returns NULL if the field has no bob object.

### See Also

**menu\_Printf, sed\_CreateBob, sed\_SetFieldBob**

### Note

This routine is implemented as a macro.

### Example

```
void bob_fkey(sed)  
    sed_type sed;  
{  
    bob_type bob;  
  
    bob = sed_GetFieldBob(sed, sed_GetFieldNo(sed));  
  
    switch( bob_Go(bob) ) {  
  
        /* ... */  
    }  
}
```

### Synopsis

```
char sed_GetFieldChar(sed, fieldno, pos);  
    sed_type sed;           the sed  
    int fieldno;            the field number  
    int pos;                the position
```

### Description

This routine returns the character at position number *pos* within field number *fieldno*'s record. The first character is at position 0.

### Return Value

The character at the specified position within the specified field's record.

### See Also

**sed\_GetChar, sed\_GetCurrChar**

### Note

This routine is implemented as a macro.

### Example

```
/* check the first letter of the first field */  
  
switch( sed_GetFieldChar(sed, 0, 0)) {  
case ' ':  
    /* ... */
```

**Synopsis**

```
int sed_GetFieldCol(sed, fieldno);  
    sed_type sed;        the sed  
    int fieldno;         field number
```

**Description**

This routine determines the display column number from the left edge of the sed (the first column is 0) of the first position in the field merge.

**Return Value**

Returns the column number of the first character in the field.

**See Also**

**sed\_GetFieldLastCol**, **sed\_GetFieldRow**, **menu\_GetFieldCol**, **menu\_GetFieldRow**

**Example**

```
int col;  
  
col = sed_GetFieldCol(sed);
```

### Synopsis

```
void sed_GetFieldColors(sed, fieldno, reg, sel);  
    sed_type sed;           the sed  
    int fieldno;           the field number  
    byte *reg;             the regular color  
    byte *sel;             the selected color
```

### Description

This routine determines the regular and selected colors for field *fieldno* and places them into the locations pointed to by *reg* and *sel*.

*reg* and *sel* must be **pointers** to bytes.

If the field is marked, the marked colors are returned.

### Return Value

There is no return value.

### See Also

**sed\_MarkField**, **sed\_SetColors**

### Example

```
byte reg, sel;  
  
sed_GetFieldColors(sed, 0, &reg, &sel);
```

## **sed\_GetFieldCount**

Get the number of fields in the sed

---

### **Synopsis**

```
int sed_GetFieldCount(sed);  
    sed_type sed;          the sed
```

### **Description**

This routine returns the number of fields in the sed.

### **Return Value**

Returns the total number of fields contained in the sed.

### **See Also**

**sed\_GetFieldNo**

### **Note**

This routine is implemented as a macro.

### **Example**

```
int i;  
  
for (i = 0; i < sed_GetFieldCount(sed); i++) {  
    /* ... */  
}
```

### Synopsis

```
VOID *sed_GetFieldData(sed, fieldno, datano);  
  
    sed_type sed;           the sed  
    int fieldno;           the field number  
    int datano;            the data number
```

### Description

This routine returns a data pointer for the field. Each field has at least one generic data pointer that the programmer can set. The data pointer is a VOID \* pointer. You can use the *datano* argument to indicate which data pointer you want (0 is the first data pointer).

A field normally has one data pointer but you can override this with the **menu\_Printf** “@fd” command when the field is defined.

The standard field functions use the field generic data pointers in the following ways:

- (0) to store border prompt strings.
- (1) to store validation data.
- (2) to store formatting information.

### Return Value

Returns the generic data pointer for the field. The generic data pointer is a VOID \* pointer. Use a type cast to convert it to a different type if necessary.

Returns NULL if *datano* exceeds the number of data pointers associated with the field.

### See Also

**menu\_Printf**, **sed\_SetFieldData**, **sed\_GetFieldDataCount**, **sed\_GetCurrFieldData**

### Note

This routine is implemented as a macro.



## Example

```
boolean int_fexit(sed)
    sed_type sed;
/*
    Validates an integer using the string in field data 1.
*/
{
    int val;

    sscanf(sed_GetCurrRecord(sed), "%d", &val);
    if ( !valid_Int(val,
        (char *) sed_GetFieldData(sed, sed_GetFieldNo(sed), 1)) ) {

        tone();
        sed_BorderPrompt(sed, "Number out of range");
        /* wait for a keystroke */
        while (!kb_Check())
            ;
        sed_BorderPrompt(sed,
            sed_GetFieldData(sed, sed_GetFieldNo(sed), 0));

        return(FALSE);
    }

    sed_DoFieldSexit(sed, sed_GetFieldNo(sed));
    sed_DoFieldSenter(sed, sed_GetFieldNo(sed));
    sed_UpdateCurrField(sed);
    sed_BorderPrompt(sed, NULL);
    return(TRUE);
}
```

## **sed\_GetFieldDataCount**

Get the field's number of data pointers

---

### **Synopsis**

```
int sed_GetFieldDataCount(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### **Description**

Returns the number of data pointers bound to the field. A field normally has one data pointer but you can override this by using the **menu\_Printf** “@fd” command when you define the field.

### **Return Value**

Returns the number of data pointers bound to the field.

### **See Also**

**menu\_Printf**, **sed\_GetFieldData**

### **Note**

This routine is implemented as a macro.

### **Example**

```
int dcount;  
  
dcount = sed_GetFieldDataCount(sed, 0);
```

**Synopsis**

```
int sed_GetFieldLastCol(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           field number
```

**Description**

This routine determines the display column number (the first column is 0) of the last position in the field merge.

**Return Value**

Returns the column number of the last character in the field.

**See Also**

**sed\_GetFieldCol, sed\_GetFieldRow**

**Example**

```
int lcol;  
  
lcol = sed_GetFieldLastCol(sed);
```

### Synopsis

```
char *sed_GetFieldName(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### Description

This routine returns the name associated with the field.

The name should be copied into a buffer if you intend to process it. The contents of the actual name string should not be altered.

Any field may be given a name. A name is a character string used to identify the field. A field may be given a name when it is created with **menu\_Printf** or it may be assigned one with **sed\_SetFieldName**.

### Return Value

Returns a pointer to the field's name string. Returns NULL if the field has no name.

### See Also

**menu\_Printf**, **sed\_GetNameNo**, **sed\_GotoNameField**, **sed\_SetFieldName**

### Note

This routine is implemented as a macro.

### Example

```
char name[22];           /* Make sure this is long enough */  
  
strcpy(name, sed_GetFieldName(sed, 3));
```

### Synopsis

```
int sed_GetFieldNo(sed);  
    sed_type sed;          the sed
```

### Description

This routine returns the current field number.

### Return Value

Returns the number of the current field.

### See Also

**sed\_DecField, sed\_GotoField, sed\_IncField**

### Note

This routine is implemented as a macro.

### Example

```
sed_GotoField(sed, sed_GetFieldNo(sed) + 2);
```

### Synopsis

```
int sed_GetFieldRow(sed, fieldno);  
    sed_type sed;          the sed  
    int fieldno;           field number
```

### Description

This routine determines the display row number from the top of the sed (the first row is 0) of the field.

### Return Value

Returns the row number of the field.

### See Also

**sed\_GetFieldCol**, **sed\_GetFieldLastCol**, **menu\_GetFieldCol**, **menu\_GetField-Row**

### Example

```
int row;  
  
row = sed_GetFieldRow(sed, 22);
```

### Synopsis

```
int sed_GetFieldWidth(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### Description

This routine returns the displayed width of field *fieldno*. The displayed width of a field is its actual width on the display. This width equals the field's merge length unless its displayed width has been explicitly set to a different value with **menu\_Printf** or **sed\_SetFieldWidth**.

A field automatically scrolls when its merge length exceeds its displayed width and an attempt is made to move the cursor past one of the field's edges.

### Return Value

Returns the displayed width of the field.

### See Also

**menu\_Printf**, **sed\_SetFieldWidth**

### Note

This routine is implemented as a macro.

### Example

```
int wid;  
  
wid = sed_GetFieldWidth(sed, 0);
```

### Synopsis

```
int sed_GetFieldXoffset(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### Description

This routine returns the value of the field's xoffset. The xoffset measures the number of record positions the field has been scrolled to the right. A value of 0 means that the field has not been scrolled.

A field automatically scrolls when its merge length exceeds its displayed width and an attempt is made to move the cursor past one of the field's edges.

### Return Value

Returns the value of the field's xoffset.

### See Also

[sed\\_SetFieldWidth](#)

### Note

This routine is implemented as a macro.

### Example

```
int xoff;  
  
xoff = sed_GetFieldXoffset(sed, 0);
```



### Synopsis

```
field_funcs_ptr sed_GetFuncs(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### Description

This routine gets the field function associated with the field.

### Return Value

A pointer to the field function structure for the specified field.

### See Also

**menu\_Printf, sed\_SetFuncs**

### Note

This routine is implemented as a macro.

**Synopsis**

```
int sed_GetGridCol(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;            the field number
```

**Description**

This routine determines the grid column number for field *fieldno*.

Each field has a position in the “grid.” The grid can be pictured as a two dimensional array of field numbers. The field numbers are sorted by location and placed into the grid when the menu is defined. The grid facilitates the operation of movement functions such as **sed\_UpField** and **sed\_LeftField**. Each field has two grid coordinates, grid row and grid column, that are used to find fields in the grid. It is possible to move to fields by grid position with **sed\_GotoGridField**.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. The grid column value for field 0 is 0. The grid column value for field 5 is 2.

0	1	2
3	4	5
6	7	8

**Return Value**

The grid column number for the field.

**See Also**

**sed\_GetGridRow**, **sed\_GotoGridField**

**Note**

This routine is implemented as a macro.

## sed\_GetGridField

Get the field number at the grid position

---

### Synopsis

```
int sed_GetGridField(sed, row, col);

    sed_type sed;           the sed
    int row;                the grid row
    int col                 the grid col
```

### Description

This routine determines the field number for grid position *row*, *col*.

Each field has a position in the “grid.” The grid can be pictured as a two dimensional array of field numbers. The field numbers are sorted by location and placed into the grid when the menu is defined. The grid facilitates the operation of movement functions such as **sed\_UpField** and **sed\_LeftField**. Each field has two grid coordinates, grid row and grid column, that are used to find fields in the grid. It is possible to move to fields by grid position with **sed\_GotoGridField**.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. The field number associated with grid position (0, 0) is 0. The field number associated with grid position (1, 2) is 5.

0	1	2
3	4	5
6	7	8

### Return Value

The field number for the grid position. Returns (-1) if there is no field at the specified position.

### See Also

**sed\_GetGridCol**, **sed\_GetGridRow**, **sed\_GotoGridField**

### Example

```
fieldno = sed_GetGridField(sed, 0, 0);
```

### Synopsis

```
int sed_GetGridRow(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;            the field number
```

### Description

This routine determines the grid row number for field *fieldno*.

Each field has a position in the “grid.” The grid can be pictured as a two dimensional array of field numbers. The field numbers are sorted by location and placed into the grid when the menu is defined. The grid facilitates the operation of movement functions such as **sed\_UpField** and **sed\_LeftField**. Each field has two grid coordinates, grid row and grid column, that are used to find fields in the grid. It is possible to move to fields by grid position with **sed\_GotoGridField**.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. The grid row value for field 0 is 0. The grid row value for field 5 is 1.

0	1	2
3	4	5
6	7	8

### Return Value

The grid row number for the field.

### See Also

**sed\_GetGridCol**, **sed\_GetGridField**, **sed\_GotoGridField**

### Example

```
int g_row;  
  
g_row = sed_GetGridRow(sed, 1);
```

### Synopsis

```
int sed_GetHeight(sed);  
    sed_type sed;           the sed
```

### Description

This routine returns the height of the sed.

### Return Value

Returns the height of the sed.

### See Also

**sed\_GetWidth, sed\_GetBorderHeight, sed\_GetMenuHeight**

### Example

```
/* don't let the sed get taller than 10 rows */  
if (sed_GetHeight(sed) > 10) {  
    sed_SetHeight(sed, 10);  
}
```

### Synopsis

```
int sed_GetLabel(sed);  
    sed_type sed;           the sed
```

### Description

This routine returns the value of the sed's label.

The label is an integer that gives a unique number to a sed. The label typically specifies a chapter number for the help system.

The label is initially set to 0.

### Return Value

The value of the sed's label.

### See Also

[sed\\_SetLabel](#)

### Note

This routine is implemented as a macro.

### Example

```
help_Show(sed_GetLabel(sed), sed_GetFieldNo(sed) + 1);
```

### Synopsis

```
menu_type sed_GetMenu(sed);  
    sed_type sed;          the sed
```

### Description

This routine returns the menu from which the sed was created.

### Return Value

Returns the menu associated with its argument.

### See Also

**menu\_Open, sed\_Open**

### Note

This routine is implemented as a macro.

### Example

```
/* ... Add a field to the sed */  
  
menu_Printf(sed_GetMenu(sed),  
    "@p[10,0]@f[#####]", &x, &int_funcs);
```

## **sed\_GetMenuHeight**

Get the height of the sed's menu

---

### **Synopsis**

```
int sed_GetMenuHeight(sed);  
    sed_type sed;           the sed
```

### **Description**

This routine returns the number of rows in the menu associated with the sed.

### **Return Value**

Returns the number of rows in the menu associated with the sed.

### **See Also**

**sed\_GetHeight, sed\_GetMenuWidth**

### **Note**

This routine is implemented as a macro.



## **sed\_GetMenuWidth**

Get the width of the sed's menu

---

### **Synopsis**

```
int sed_GetMenuWidth(sed);  
    sed_type sed;           the sed
```

### **Description**

This routine returns the number of columns in the menu associated with the sed. The widest row in the menu determines this number.

### **Return Value**

Returns the number of columns in the menu associated with the sed.

### **See Also**

**sed\_GetWidth, sed\_GetMenuHeight**

### **Note**

This routine is implemented as a macro.

### Synopsis

```
char *sed_GetMerge(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;            field number
```

### Description

This routine returns the merge of the field specified by the field number *fieldno*. A merge is the string containing the entire contents of the field, both the writeable and nonwriteable positions.

You should copy the merge into a buffer before you use it. The contents of the actual merge should not be altered.

### Return Value

Returns a pointer to the merge string of the specified field.

### See Also

**sed\_GetCurrMerge, sed\_GetRecord, sed\_GetScratchPad**

### Note

This routine is implemented as a macro.

### Example

```
int fld = 0;  
char buffer[101]; /* make sure this is long enough */  
  
strcpy(buffer, sed_GetMerge(sed, fld));
```

## sed\_GetMergeLen

Get the length of the field's merge

---

### Synopsis

```
int sed_GetMergeLen(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           field number
```

### Description

This routine returns the length of the merge string of field *fieldno*. The merge consists of both the nonwriteable and writeable positions in the field.

### Return Value

Returns the length of the specified field's merge string.

### See Also

**sed\_GetMerge**

### Note

This routine is implemented as a macro.

### Example

```
int mlen;  
  
mlen = sed_GetMergeLen(sed, 2);
```

### Synopsis

```
int sed_GetMergePos(sed);  
    sed_type sed;          the sed
```

### Description

This routine returns the current position in the record. Note that the merge position is calculated using both the writeable and nonwriteable positions.

### Return Value

Returns the current position in the merge.

### See Also

**sed\_GetRecordPos**

### Note

This routine is implemented as a macro.

### Example

```
int pos;  
  
pos = sed_GetMergePos(sed);
```

## sed\_GetNameNo

Get the field number associated with a name

---

### Synopsis

```
int sed_GetNameNo(sed, name);  
  
    sed_type sed;           the sed  
    char *name;             the name
```

### Description

This routine returns the field number of the first field with name *name*. If no field has the given name, it returns -1. *name* must match a field name exactly.

### Return Value

Returns the field number of the field with the given name or -1 if no field has the given name.

### See Also

`menu_Printf`, `sed_GetFieldName`, `sed_GotoNameField`, `sed_SetFieldName`, `sed_GetNameVar`

### Note

This routine is implemented as a macro.

### Example

```
int fieldno;  
  
fieldno = sed_GetNameNo(sed, "total");
```

**Synopsis**

```
VOID *sed_GetNameVar(sed, name);  
  
    sed_type sed;           the sed  
    char *name;            field name
```

**Description**

This routine returns the pointer to the variable for the field with name *name*. If no field has the given name, it returns NULL. *name* must match a field name exactly.

This routine is commonly used with screen files.

**Return Value**

This routine returns a pointer to the variable for the field with field number *fieldno*. The variable is a VOID \* pointer. Use a type cast to convert it to another type when necessary. If it cannot find the field name it returns NULL.

**See Also**

**sed\_GetVar, sed\_GetNameNo**

**Example**

```
/* ... */  
  
/* Load screen from file, allocate field variables */  
sed = sfile_LoadSed(sfile, "screen1", SED_ALLOC);  
  
/* Set field variables */  
strcpy((char *) sed_GetNameVar(sed, "animal"), animal);  
  
/* ... */
```

### Synopsis

```
void sed_GetPosition(sed, row, col);  
    sed_type sed;           the sed  
    int *row, *col;         the position of the sed
```

### Description

This routine returns the position of the upper left hand corner of the sed into the locations pointed to by *row* and *col*. If there is a border attached to the sed and its upper left hand corner is above that of the sed, **sed\_GetPosition** returns the position of the border.

### Return Value

There is no return value.

### See Also

**sed\_SetPosition**, **sed\_GetCorners**

### Example

```
int row, col;  
  
sed_GetPosition(sed, &row, &col);
```

### Synopsis

```
char *sed_GetRecord(sed, fieldno);  
    sed_type sed;          the sed  
    int fieldno;           field number
```

### Description

This routine returns the record of the field specified by the field number *fieldno*. A record consists of only the writeable positions in the field.

### Return Value

Returns a pointer to the record string of the specified field.

### See Also

**sed\_SetRecord, sed\_GetCurrRecord, sed\_GetMerge**

### Note

This routine is implemented as a macro.

### Example

```
void string_sexit(sed, fieldno)  
    sed_type sed;  
    int fieldno;  
/*  
    Copy the record string back into the native string.  
*/  
{  
    strcpy((char *) sed_GetVar(sed, fieldno),  
           sed_GetRecord(sed, fieldno));  
}
```



### Synopsis

```
int sed_GetRecordLen(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           field number
```

### Description

This routine returns the length of the record string of the field specified by field number *fieldno*. A record consists of only the writeable positions in the field.

### Return Value

Returns the length of the specified field's record string.

### See Also

**sed\_GetRecord, sed\_GetCurrRecordLen**

### Note

This routine is implemented as a macro.

### Example

```
void long_senter(sed, fieldno)  
    sed_type sed;  
    int fieldno;  
/*  
    Convert native type to string for record.  
*/  
{  
    char *s;  
  
    s = sed_GetScratchPad(sed);  
    sprintf(s, "%ld", *((long *) sed_GetVar(sed, fieldno)));  
  
    strright(s, sed_GetRecordLen(sed, fieldno));  
    sed_SetRecord(sed, s, fieldno);  
}
```

### Synopsis

```
int sed_GetRecordPos(sed);  
    sed_type sed;           the sed
```

### Description

This routine returns the current position in the current record. Note that the record position is calculated using only the writeable positions and that the record starts with position 0.

### Return Value

Returns the current position in the current record.

### See Also

**sed\_DecChar, sed\_GoEnd, sed\_GoHome, sed\_GotoChar, sed\_IncChar**

### Note

This routine is implemented as a macro.

### Example

```
if (sed_GetRecordPos(sed) > 2) {  
    /* ... */  
}
```

### Synopsis

```
char *sed_GetScratchPad(sed);  
    sed_type sed;          the sed;
```

### Description

This routine returns a pointer to the sed's scratch pad. The scratch pad is a buffer used by **menu\_Printf** to create fields. Its length exceeds the length of the sed's longest field. The scratch pad is used for field operations where a buffer whose size depends on the field's size is necessary. For example, consider a field **senter** function that centers a string within the field's record:

```
void center_senter(sed, fld)  
    sed_type sed;  
    int fld;  
{  
    /* copy field variable into scratch pad */  
    strcpy(sed_GetScratchPad(sed), (char *) sed_GetVar(sed, fld));  
  
    /* center the string within the scratch pad */  
    strcenter(sed_GetScratchPad(sed), sed_GetRecordLen(sed, fld));  
  
    /* copy the centered string into the field's record */  
    sed_SetRecord(sed, sed_GetScratchPad(sed), fld);  
}
```

The above routine will work regardless of the length of the field.

You should only use the scratch pad should to hold data temporarily as it is used by other routines, especially field functions.

### Return Value

Returns a pointer to the scratch pad buffer.

### See Also

**sed\_GetScratchSize**

## Note

This routine is implemented as a macro.

## Example

```
void long_senter(sed, fieldno)
    sed_type sed;
    int fieldno;
/*
    Convert native type to string for record.
*/
{
    char *s;

    /* hold string in scratch pad */
    s = sed_GetScratchPad(sed);

    sprintf(s, "%ld", *((long *) sed_GetVar(sed, fieldno)));

    strright(s, sed_GetRecordLen(sed, fieldno));
    sed_SetRecord(sed, s, fieldno);
}
```

## **sed\_GetScratchSize**

Get the size of the sed's scratch pad

---

### **Synopsis**

```
int sed_GetScratchSize(sed);  
    sed_type sed;          the sed
```

### **Description**

This routine returns the length of the sed's scratch pad buffer in characters. The length of the scratch pad exceeds the length of the sed's longest field.

### **Return Value**

Returns the length of the sed's scratch pad.

### **See Also**

**sed\_GetScratchPad**

### **Note**

This routine is implemented as a macro.

### Synopsis

```
void sed_GetSize(sed, height, width);  
  
    sed_type sed;           the sed  
    int *height;            the height of the sed  
    int *width;             the width of the sed
```

### Description

This routine returns the size of the sed in the locations pointed to by *height* and *width*.

### Return Value

There is no return value.

### See Also

**sed\_GetHeight, sed\_GetWidth**

### Example

```
int hgt, wid;  
  
sed_GetSize(sed, &hgt, &wid);
```

### Synopsis

```
unsigned int sed_GetTB(sed, buffer, len, mode);  
  
sed_type sed;           the sed  
char *buffer;           buffer to copy text into  
unsigned int len;        length of the buffer  
int mode;                newline mode
```

### Description

This routine copies text from the sed's text buffer into *buffer*. It copies up to *len* characters starting at the current text cursor position. The text cursor is advanced by the number of characters read. This makes it possible to call **sed\_GetTB** repeatedly to read large text buffers.

**sed\_RewindTB** can be called prior to calling **sed\_GetTB** to move the text cursor to the start of the text buffer.

*mode* determines how the newlines are treated. It can have one of the following values:

**TED\_HARD** Get the text as it appears on the display with '\n' characters at the end of each line.

**TED\_SOFT** Get the text as it is stored in the text buffer with '\n' characters only where they actually exist.

A terminating '\0' is placed in *buffer* after the last character read. The length of *buffer* should be at least one longer than *len* in order to accommodate the terminating '\0'.

### Return Value

Returns the number of characters actually copied into the text buffer. This number can be less than *len* if **sed\_GetTB** reaches the end of the text buffer.

### Note

This routine is implemented as a macro.

## See Also

**sed\_SetTB, sed\_RewindTB**

## Example

```
char buffer[BUFLen];
unsigned count;

/* read the text buffer and print it out */
sed_RewindTB(sed);

do {
    count = sed_GetTB(sed, buffer, BUFLen, TED_SOFT);
    printf(buffer);
} while (count == BUFLen);
```



### Synopsis

```
VOID *sed_GetVar(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           field number
```

### Description

This routine returns the pointer to the variable for field *fieldno*. The pointer is bound to the field during the calls to **menu\_Printf**.

### Return Value

This routine returns a pointer to the variable for the field with field number *fieldno*. The variable is a VOID \* pointer. Use a type cast to convert it to another type when necessary.

### See Also

**sed\_GetCurrVar**, **sed\_GetNameVar**

### Note

This routine is implemented as a macro.

### Example

```
void money_senter(sed, fieldno)  
    sed_type sed;  
    int fieldno;  
/*  
    Convert native type to string for record.  
*/  
{  
    char *s;  
  
    s = sed_GetScratchPad(sed);  
    sprintf(s, "%ld", *((long *) sed_GetVar(sed, fieldno)));  
    strright(s, sed_GetRecordLen(sed, fieldno));  
    sed_SetRecord(sed, strdec(s, DECP), fieldno);  
}
```

**Synopsis**

```
SIZE_T sed_GetVarSize(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

**Description**

This routine returns the size of the field's variable. This size is defined in the field function structure.

The field's variable size is used by sleds (scrolling seds) and screen files to allocate storage space for field variables.

**Return Value**

Returns the size of the field's variable or one of the special values described above.

Note that `SIZE_T` is equivalent to `size_t` on ANSI C compilers and equivalent to unsigned int on older compilers..

**Note**

This routine is implemented as a macro.

## sed\_GetWidth

Get the sed's width

---

### Synopsis

```
int sed_GetWidth(sed);  
    sed_type sed;           the sed
```

### Description

This routine returns the width of the sed.

### Return Value

Returns the width of the sed.

### See Also

**sed\_GetHeight, sed\_GetBorderWidth, sed\_GetMenuWidth**

### Example

```
if (sed_GetWidth(sed) > 40) {  
    sed_SetWidth(sed, 40);  
}
```

### Synopsis

```
int sed_GetXoffset(sed);  
    sed_type sed;           the sed
```

### Description

This routine returns the value of the sed's xoffset. The xoffset measures the number of columns the sed has been scrolled to the right. A value of 0 implies that the sed has not been scrolled horizontally.

### Return Value

Returns the value of the xoffset; 0 if the sed has not been scrolled horizontally.

### See Also

**sed\_GetYoffset**

### Note

This routine is implemented as a macro.

### Example

```
switch(kb_Read()) {  
case CTRL_LEFT:  
    sed_ScrollLeft(sed, sed_GetXoffset(sed));  
    break;  
/* ... */
```

### Synopsis

```
int sed_GetYoffset(sed);  
    sed_type sed;           the sed
```

### Description

This routine returns the value of the sed's yoffset. The yoffset measures the number of rows the sed has been scrolled down. A value of 0 implies that the sed has not been scrolled vertically.

### Return Value

Returns the value of the yoffset; 0 if the sed has not been scrolled vertically.

### See Also

**sed\_GetXoffset**

### Note

This routine is implemented as a macro.

### Example

```
switch(kb_Read()) {  
case HOME:  
    sed_ScrollUp(sed, sed_GetYoffset(sed));  
    break;  
/* ... */
```

**Synopsis**

```
int sed_Go(sed);  
    sed_type sed;           the sed
```

**Description**

This routine gets input from the user until the user decides to exit. It does not repaint the sed but it does highlight the current field. It returns the value of the baton. It saves the cursor type upon entry and restores it upon exit.

In detail, the **sed\_Go** routine does the following. **sed\_Go** first checks if the current field is protected. If it is protected, it makes the next unprotected field the current field. Next it saves the cursor type, sets the sed's active mode, and highlights the current field. Then **sed\_Go** calls the **fenter** function for the current field. It then calls **fkey** function repeatedly until the **fkey** function calls **sed\_ToggleExit**. After **fkey** calls **sed\_ToggleExit**, **sed\_Go** calls the **fexit** function for the current field. If the **fexit** function returns FALSE, **sed\_Go** continues calling the **fkey** function. Otherwise, if the **fexit** function returns TRUE, **sed\_Go** calls the **sexit** function for each field in the sed, resets the active mode, unhighlights the current field, restores the cursor type, and returns the value of the baton.

**sed\_Go** restores the cursor type when it passes control from one field to another.

If a sed contains no fields or if it finds no unprotected fields, **sed\_Go** returns immediately.

If you set a *nextwin* with **sed\_SetNextWin**, **sed\_Go** will pass control to it instead of returning.

For an explanation and possible uses of the baton, see **sed\_SetBaton**.

See **sed\_ProtectField** for an explanation of protected fields.

See **sed\_IsActive** for an explanation of active seds.

Refer to the chapter "Seds" in the *C-scape Manual* for more information.

## Return Value

Returns the value of the baton which is normally the field number plus one from which the user exited the sed or 0 (zero) if the user presses **Esc**.

## Note

This routine is implemented as a macro.

## See Also

**menu\_Open, menu\_Printf, sed\_Open, sed\_Repaint, sed\_Close, sed\_GetBaton, sed\_IsActive, sed\_ProtectField, sed\_SetBaton, sed\_SetNextWin, sed\_ToggleExit**

## Example

```
/* ... */

sed = sed_Open(menu);
sed_SetColors(sed, 0x07, 0x07, 0x70);
sed_SetPosition(sed, 8, 19);

sed_Repaint(sed);
ret = sed_Go(sed);
sed_Pop(sed);

/* ... */
```

**Synopsis**

```
void sed_GoEnd(sed);  
    sed_type sed;           the sed
```

**Description**

This routine moves the cursor to the position at the end of the record string in the current field.

**Return Value**

There is no return value.

**See Also**

**sed\_DecChar, sed\_GoHome, sed\_GotoChar, sed\_IncChar**

**Example**

```
void string_fkey(sed)  
    sed_type sed;  
{  
    int scancode;  
  
    switch (scancode = kb_Read()) {  
    case HOME:  
        /* move the cursor to the beginning of the field */  
        sed_GoHome(sed);  
        break;  
    case END:  
        /* move the cursor to the end of the record string */  
        sed_GoEnd(sed);  
        break;  
    /* .... */
```



## sed\_GoHome

Go to the start of the current field

---

### Synopsis

```
void sed_GoHome(sed);  
    sed_type sed;           the sed
```

### Description

This routine moves the cursor to the first position in the current field.

### Return Value

There is no return value.

### See Also

**sed\_DecChar, sed\_GoEnd, sed\_GotoChar, sed\_IncChar**

### Example

```
void string_fkey(sed)  
{  
    sed_type sed;  
    int scancode;  
  
    switch (scancode = kb_Read()) {  
    case HOME:  
        /* move the cursor to the beginning of the field */  
        sed_GoHome(sed);  
        break;  
    case END:  
        /* move the cursor to the end of the record string */  
        sed_GoEnd(sed);  
        break;  
  
        /* .... */  
    }  
}
```

### Synopsis

```
void sed_GotoChar(sed, position);  
    sed_type sed;      the sed  
    int position;      the new position
```

### Description

This routine moves the cursor to the specified record position in the current field.

### Return Value

There is no return value.

### See Also

**sed\_DecChar, sed\_GoEnd, sed\_GoHome, sed\_IncChar**

### Example

```
void phone_fenter(sed)  
    sed_type sed;  
/*  
    Move the cursor past the area code...  
*/  
{  
    sed_GotoChar(sed, 3);  
}
```

### Synopsis

```
int sed_GotoField(sed, fieldno);  
  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### Description

This routine makes the field *fieldno* the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_GotoField** does the following: If the sed is in active mode, it calls the **fexit** function for the current field. If **fexit** returns FALSE, it does nothing and returns SED\_INVALID. If the new field is protected the routine does nothing and returns SED\_STUCK. Otherwise, field *fieldno* becomes the new current field. Before **sed\_GotoField** enters the new field, it calls the **fenter** function for that field.

You must not call this routine in either a field's **fenter** or **fexit** routines.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_GotoField** does not call the **fexit** and **fenter** functions.

### Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if the new field is protected.

### See Also

#### sed\_ProtectField

### Example

```
switch( ascii( kb_Read() ) ) {  
case '0':  
    sed_GotoField(sed, 0);  
    break;  
case '1':  
    sed_GotoField(sed, 1);  
    break;
```

### Synopsis

```
int sed_GotoFirstField(sed);  
    sed_type sed;          the sed
```

### Description

This routine makes the first field in the sed the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_GotoFirstField** does the following: If the sed is in active mode it calls the **fexit** function for the current field. If **fexit** returns FALSE, this routine does nothing and returns SED\_INVALID. If the current field is the first unprotected field the routine does nothing and returns SED\_STUCK. Otherwise, the first unprotected field becomes the new current field. Before it enters the new field, **sed\_GotoFirstField** calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_GotoFirstField** does not call the **fexit** and **fenter** functions.

You must not call this routine in either a field's **fenter** or **fexit** routines.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. A "p" following the number represents a protected field. Calling **sed\_GotoFirstField** from field 4 would make field 1 the new current field (field 0 is protected).

0p	1	2
3	4	5
6	7	8

### Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it could not find an unprotected new field to which it can move.

## See Also

**sed\_GotoLastField, sed\_ProtectField**

## Example

```
switch(kb_Read()) {
case DOWN:
    if (sed_IncField(sed) == SED_STUCK) {
        sed_GotoFirstField(sed);
    }
    break;
/* ... */
```

### Synopsis

```
int sed_GotoGridField(sed, row, col);  
  
    sed_type sed;           the sed  
    int row;                the grid row  
    int col;                the grid column
```

### Description

This routine takes the field specified by the grid coordinates and makes it the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_GotoGridField** does the following: If the sed is in active mode it calls the **fexit** function for the current field. If **fexit** returns FALSE, this routine does nothing and returns SED\_INVALID. If the specified field is protected or if the row and column do not specify a valid field the routine does nothing and returns SED\_STUCK. Otherwise, the specified field becomes the new current field. Before **sed\_GotoGridField** it enters the new field, it calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_GotoGridField** does not call the **fexit** and **fenter** functions are not called.

Each field is assigned a position in the “grid.” The grid can be pictured as a two dimensional array of field numbers. The field numbers are sorted by location and placed into the grid when the menu is defined. The grid is used to facilitate the operation of movement functions such as **sed\_UpField** and **sed\_LeftField**. Each field has two grid coordinates, grid row and grid column, that are used to find fields in the grid. It is possible to determine a field’s grid coordinates using **sed\_GetGridCol** and **sed\_GetGridRow**.

You must not call this routine in either a field’s **fenter** or **fexit** routines.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. **sed\_GotoGridField**(sed, 1, 1) would make field 4 the new current field.

0	1	2
3	4	5
6	7	8

## Return Value

Returns **SED\_MOVED** if successful, **SED\_INVALID** if the original field's **fxit** function failed, and **SED\_STUCK** if it could not find an unprotected new field to which it can move.

## See Also

**sed\_GetGridCol**, **sed\_GetGridField**, **sed\_GetGridRow**

## Example

```
switch(kb_Read()) {  
  case HOME:  
    sed_GotoGridField(sed, 0, 0);  
    break;  
  /* ... */  
}
```

### Synopsis

```
int sed_GotoLastField(sed);  
    sed_type sed;          the sed
```

### Description

This routine makes the last field in the sed the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_GotoLastField** does the following: If the sed is in active mode it calls the **fexit** function for the current field. If **fexit** returns FALSE, it does nothing and returns SED\_INVALID. If the current field is the last unprotected field the routine does nothing and returns SED\_STUCK. Otherwise, the last unprotected field becomes the new current field. Before it enters the new field, it calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_GotoLastField** does not call the **fexit** and **fenter** functions.

You must not call this routine in either a field's **fenter** or **fexit** routines.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. A "p" following the number represents a protected field. Calling **sed\_GotoLastField** from field 4 would make field 7 the new current field (field 8 is protected).

0	1	2
3	4	5
6	7	8p



## Return Value

Returns `SED_MOVED` if successful, `SED_INVALID` if the original field's **fixit** function failed, and `SED_STUCK` if it could not find an unprotected new field to which it can move.

## See Also

**`sed_GotoFirstField`, `sed_ProtectField`**

## Example

```
switch(kb_Read()) {
case UP:
    if (sed_DecField(sed) == SED_STUCK) {
        sed_GotoLastField(sed);
    }
    break;
/* ... */
```

### Synopsis

```
int sed_GotoNameField(sed, name);  
    sed_type sed;           the sed  
    char *name;             the field name
```

### Description

This routine makes the field with name *name* the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_GotoNameField** does the following: If the sed is in active mode it calls the **fexit** function for the current field. If **fexit** returns FALSE, it does nothing and returns SED\_INVALID. If the new field is protected or if there is no field with the name *name* the routine does nothing and returns SED\_STUCK. Otherwise, the routine changes the current field to the new field. Before it enters the new field, it calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_GotoNameField** does not call the **fexit** and **fenter** functions. You must not call this routine in either a field's **fenter** or **fexit** routines.

Any field may be given a name. A name is a character string used to identify the field. A field may be given a name when it is created with **menu\_Printf** or it may be assigned one with **sed\_SetFieldName**. It is possible to determine a field's name using **sed\_GetFieldName**.

### Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it could not find an unprotected field with the given name. *name* must match the field's name exactly.

### See Also

**menu\_Printf**, **sed\_SetFieldName**

### Example

```
sed_GotoNameField(sed, "Total");
```

**Synopsis**

```
boolean sed_IncChar(sed);  
    sed_type sed;           the sed
```

**Description**

This routine moves the cursor to the next character in a field. If the cursor is already at the last position in the field, the routine does nothing.

**Return Value**

This routine returns TRUE if it succeeded in moving the cursor, otherwise it returns FALSE.

**See Also**

**sed\_GetRecordPos, sed\_GetMergePos, sed\_GoEnd, sed\_GoHome, sed\_Go-  
toChar, sed\_DecChar**

**Example**

```
void string_fkey(sed)  
    sed_type sed;  
{  
    int scancode;  
  
    switch (scancode = kb_Read()) {  
    case LEFT:  
        /* move the cursor backwards if the left arrow is pressed */  
        sed_DecChar(sed);  
        break;  
    case RIGHT:  
        /* move the cursor forwards if the right arrow is pressed */  
        sed_IncChar(sed);  
        break;  
  
        /* .... */  
    }  
}
```

### Synopsis

```
int sed_IncField(sed);  
    sed_type sed;          the sed
```

### Description

This routine takes the field after the current field and makes it the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_IncField** does the following. If the sed is in active mode, it calls the **fexit** function for the current field. If **fexit** returns FALSE, this routine does nothing and returns SED\_INVALID. If the current field is the last unprotected field the routine does nothing and returns SED\_STUCK. Otherwise, the next field becomes the new current field. Before it enters the new field, **sed\_IncField** calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_IncField** does not call the **fexit** and **fenter** functions.

You must not call this routine in either a field's **fenter** or **fexit** routines.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. A "p" following the number represents a protected field. Calling **sed\_IncField** from field 4 would make field 5 the new current field. Calling **sed\_IncField** from field 1 would make field 3 the new current field (field 2 is protected).

0	1	2p
3	4	5
6	7	8

## Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it could not find an unprotected new field to which it can move.

## See Also

**sed\_GetFieldNo, sed\_GotoField, sed\_DecField, sed\_ProtectField**

## Example

```
boolean inter_field(sed, scancode)
    sed_type sed;
    int scancode;

{
    switch (scancode) {
        /* ... */
        case UP:
            sed_DecField(sed);
            return(TRUE);
        case DOWN:
            sed_IncField(sed);
            return(TRUE);
        default:
            break;
    }
    return(FALSE);
}
```

### Synopsis

```
boolean sed_InsertRows(sed, row, count);  
    sed_type sed;           the sed  
    int row;                row to insert at  
    int count;              number of rows to insert
```

### Description

This routine inserts *count* rows from the sed starting at menu row *row*. The new rows are blank and contain no text or fields.

### Return Value

Returns TRUE if successful and FALSE if unable to insert the rows.

### See Also

[sed\\_DeleteRows](#)

### Example

```
sed_InsertRows(sed, 2, 2);
```

**Synopsis**

```
boolean sed_IsActive(sed);  
    sed_type sed;          the sed
```

**Description**

This routine determines if the sed is in active mode. The active mode is set to TRUE upon entering **sed\_Go** and set back to FALSE upon leaving **sed\_Go**.

When the sed is active, the field movement functions (**sed\_IncField**, etc.) call the **fenter** and **fexit** functions, highlight the current field, and move the cursor.

When the sed is inactive, the field movement functions skip the **fenter** and **fexit** functions, paint the current field in its regular color, and ignore the cursor.

**Return Value**

Returns TRUE if the sed is in active mode and FALSE otherwise.

**See Also**

**sed\_Go**

**Note**

This routine is implemented as a macro.

**Example**

```
if (sed_IsActive(sed)) {  
    /* ... */  
}
```

### Synopsis

```
boolean sed_IsEnd(sed);  
    sed_type sed;          the sed
```

### Description

This routine determines if the current record position is at the end of the current field.

### Return Value

Returns a boolean value. The value is TRUE if the current record position is the end of the current field.

### See Also

**sed\_IsHome**

### Note

This routine is implemented as a macro.

### Example

```
if (sed_IsEnd(sed)) {  
    /* ... */  
}
```



### Synopsis

```
char *sed_IsFieldName(sed, fld, name);  
  
    sed_type sed;           the sed  
    int fld;                the field number  
    char *name;             the name of the field
```

### Description

This function checks if the field *fld* in the sed *sed* has the name *name*. If the so, the function returns TRUE; otherwise, FALSE.

Any field may be given a name. A name is a character string used to identify the field. A field may be given a name when it is created with **menu\_Printf** or it may be assigned one with **sed\_SetFieldName**.

### Return Value

Returns TRUE if field *fld* has the name *name*; otherwise, FALSE.

### See Also

**menu\_Printf**, **sed\_GetNameNo**, **sed\_GotoNameField**, **sed\_SetFieldName**

### Example

```
if (sed_IsFieldName(sed, sed_GetFieldNo(sed), "total")) {  
    /* current field is the "total" field */  
    /* ... */  
}
```

## **sed\_IsHome**

Check if the cursor is at the start of a field

---

### **Synopsis**

```
boolean sed_IsHome(sed);  
    sed_type sed;           the sed
```

### **Description**

This routine determines if the current record position is at the start of the current field.

### **Return Value**

Returns a boolean value. The value is TRUE if the current record position is the start of the current field.

### **See Also**

**sed\_IsEnd**

### **Note**

This routine is implemented as a macro.

### **Example**

```
if (sed_IsHome(sed)) {  
    /* ... */  
}
```

**Synopsis**

```
boolean sed_IsMarkedField(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the fieldno
```

**Description**

This routine determines if field *fieldno* has been marked with **sed\_MarkField**.

**Return Value**

Returns TRUE if the specified field is marked and FALSE otherwise.

**See Also**

**sed\_MarkField**, **sed\_UnMarkField**

**Note**

This routine is implemented as a macro.

**Example**

```
if (sed_IsMarkedField(sed, 0)) {  
    /* ... */  
}
```

### Synopsis

```
boolean sed_IsProtectedField(sed, fieldno);  
  
    sed_type sed;           the sed  
    int fieldno;           the fieldno
```

### Description

This routine determines if field *fieldno* is protected.

### Return Value

Returns TRUE if the field is protected and FALSE otherwise.

### See Also

**menu\_Printf, sed\_ProtectField, sed\_UnProtectField**

### Note

This routine is implemented as a macro.

### Example

```
if (sed_IsProtectedField(sed, 2)) {  
    /* ... */  
}
```

### Synopsis

```
int sed_LeftField(sed);  
    sed_type sed;           the sed
```

### Description

This routine takes the field to the left of current field and makes it the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_LeftField** does the following. If the sed is in active mode, it calls the **fexit** function for the current field. If **fexit** returns FALSE, it does nothing and returns SED\_INVALID. If the current field is the left-most unprotected field the routine does nothing and returns SED\_STUCK. Otherwise, the field to the left becomes the new current field. Before it enters the new field, **sed\_LeftField** calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_LeftField** does not call the **fexit** and **fenter** functions.

You must not call this routine in either a field's **fenter** or **fexit** routines.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. A “p” following the number represents a protected field. Calling **sed\_LeftField** from field 4 would make field 3 the new current field. Calling **sed\_LeftField** from field 2 would make field 0 the new current field (field 1 is protected).

0	1p	2
3	4	5
6	7	8

## Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it could not find an unprotected new field to which it can move.

## See Also

**sed\_UpField, sed\_RightField, sed\_DownField, sed\_ProtectField**

## Note

This routine is implemented as a macro.

## Example

```
switch (scancode) {
/* ... */
  case UP:
    sed_UpField(sed);
    return(TRUE);
  case DOWN:
    sed_DownField(sed);
    return(TRUE);
  case LEFT:
    sed_LeftField(sed);
    return(TRUE);
  case RIGHT:
    sed_RightField(sed);
    return(TRUE);
/* ... */
```

### Synopsis

```
void sed_MarkField(sed, fieldno, reg, sel);  
  
    sed_type sed;           the sed  
    int fieldno;           the fieldno  
    byte reg;              regular color of the field  
    byte sel;              selected color of the field
```

### Description

This routine marks field *fieldno* with the regular attribute *reg*. When the field is selected, the field is colored with the selected attribute *sel*.

### Return Value

There is no return value.

### Note

This routine is implemented as a macro.

### See Also

**sed\_IsMarked, sed\_UnMarkField**

### Example

```
sed_MarkField(sed, 4, 0x02, 0x20);  
sed_UpdateField(sed, 4);
```

### Synopsis

```
void sed_MoveField(sed, fieldno, row, col);  
    sed_type sed;           the sed  
    int fieldno;           the field number  
    int row;               new field row  
    int col;               new field column
```

### Description

This routine moves a field to a new position in the sed's menu. The new location is given in menu coordinates.

**sed\_MoveField** does not repaint the sed to show the field in its new position. Use either **sed\_Repaint** or **sed\_Update** for this purpose.

### Return Value

There is no return value.

### See Also

#### **sed\_SwapFields**

### Example

```
/* Move field 1 to a new location */  
  
sed_MoveField(sed, 1, rand() % disp_GetHeight(),  
              rand() % disp_GetWidth());
```



## sed\_Ok

Check if the sed is valid

---

### Synopsis

```
boolean sed_Ok(sed);  
sed_type sed;           the sed
```

### Description

This routine checks if a sed is valid. It is not for checking if a sed has been closed, but rather to ascertain if a particular sed's integrity has been violated.

### Return Value

Returns TRUE if the sed is intact; FALSE, otherwise.

### Example

```
if (!sed_Ok(sed)) {  
    exit(1);  
}
```

### Synopsis

```
sed_type sed_Open(menu);  
menu_type menu;          the menu
```

### Description

This routine creates a new sed object and returns a handle to it. If there is not enough memory to open a new sed object, the routine returns NULL.

The sed object can be considered the dynamic portion of a screen since it contains all the dynamic information of a screen such as positioning, field contents, etc.

You can create only one sed from a particular menu.

### Return Value

Returns a handle to the new sed object. A NULL pointer value indicates insufficient memory.

### Note

This routine is implemented as a macro.

### See Also

**menu\_Open, sed\_Close**

## Example

```
menu_type menu;
sed_type sed;

menu = menu_Open();
menu_Printf(menu, "Yodel Ho, kidz!");
menu_Flush(menu);

if ((sed = sed_Open(menu)) == NULL) {
    menu_Destroy(menu);
    return(0);
}

/* ... */

sed_Close(sed);
```

### Synopsis

```
char sed_Overwrite(sed, c);  
    sed_type sed;          the sed  
    int c;                 new character to overwrite
```

### Description

This routine overwrites the character in the current field at the current position with character *c*.

In the diagram below, the field is enclosed in a box and the underscore represents the current cursor position.

```
sed_Overwrite(sed, 'l');
```

has the following effect:

before

hex <u>l</u> o mom
--------------------

after:

he <u>l</u> lo mom
--------------------

----> returns('x')

### Return Value

Returns the value of the character over which it wrote.

### See Also

**sed\_PullLeft, sed\_PullRight, sed\_PushLeft, sed\_PushRight**

## Example

```
void string_fkey(sed)
    sed_type sed;
{
    int  scancode, key;

    scancode = kb_Read();

    switch(scancode) {

        /* ... */
    default:
        key = ascii(scancode);
        if (isprint(key)) {
            if (kb_Insert()) {
                sed_PushRight(sed, key);
            }
            else {
                sed_Overwrite(sed, key);
            }
            sed_IncChar(sed);
        }
        break;
    }
}
```

### Synopsis

```
int sed_PageDown(sed);  
    sed_type sed;           the sed
```

### Description

This routine scrolls the sed down one page, if possible. If the scroll is successful the routine attempts to move to a field on the new page that corresponds to the position of the current field in the current page.

If it is not possible to move down a page the routine returns SED\_STUCK.

If the sed is in active mode, **sed\_PageDown** calls the **fexit** function for the current field. If **fexit** returns FALSE, this routine does nothing and returns SED\_INVALID. If there are no fields on the new page, the current field remains the same (though it is not visible). Otherwise, the routine changes the current field to the new field. Before it enters the new field, it calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_PageDown** does not call the **fexit** and **fenter** functions.

### Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it is not possible to move down a page.

### See Also

**sed\_PageLeft**, **sed\_PageRight**, **sed\_PageUp**

### Example

```
switch (scancode) {  
case PGUP:  
    sed_PageUp(sed);  
    return(TRUE);  
case PGDN:  
    sed_PageDown(sed);  
    return(TRUE);  
/* ... */
```

### Synopsis

```
int sed_PageLeft(sed);  
    sed_type sed;          the sed
```

### Description

This routine scrolls the sed left one page, if possible. If the scroll is successful the routine attempts to move to a field on the new page that corresponds to the position of the current field in the current page.

If it is not possible to move left a page the routine returns SED\_STUCK.

If the sed is in active mode, **sed\_PageLeft** calls the **fexit** function for the current field. If **fexit** returns FALSE, this routine does nothing and returns SED\_INVALID. If there are no fields on the new page, the current field remains the same (though it is not visible). Otherwise, the routine changes the current field to the new field. Before it enters the new field, it calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_PageLeft** does not call the **fexit** and **fenter** functions.

### Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it is not possible to move left a page.

### See Also

**sed\_PageDown**, **sed\_PageRight**, **sed\_PageUp**

### Example

```
switch (scancode) {  
case CTRL_LEFT:  
    sed_PageLeft(sed);  
    return(TRUE);  
case CTRL_RIGHT:  
    sed_PageRight(sed);  
    return(TRUE);  
/* ... */
```

### Synopsis

```
int sed_PageRight(sed);  
    sed_type sed;          the sed
```

### Description

This routine scrolls the sed left one page, if possible. If the scroll is successful the routine attempts to move to a field on the new page that corresponds to the position of the current field in the current page.

If it is not possible to move right a page the routine returns `SED_STUCK`.

If the sed is in active mode, **sed\_PageRight** calls the **fexit** function for the current field. If **fexit** returns `FALSE`, this routine does nothing and returns `SED_INVALID`. If there are no fields on the new page, the current field remains the same (though it is not visible). Otherwise, the routine changes the current field to the new field. Before it enters the new field, it calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_PageRight** does not call the **fexit** and **fenter** functions.

### Return Value

Returns `SED_MOVED` if successful, `SED_INVALID` if the original field's **fexit** function failed, and `SED_STUCK` if it is not possible to move right a page.

### See Also

**sed\_PageDown**, **sed\_PageLeft**, **sed\_PageUp**

### Example

```
switch (scancode) {  
case CTRL_LEFT:  
    sed_PageLeft(sed);  
    return(TRUE);  
case CTRL_RIGHT:  
    sed_PageRight(sed);  
    return(TRUE);  
/* ... */
```



### Synopsis

```
int sed_PageUp(sed);  
    sed_type sed;           the sed
```

### Description

This routine scrolls the sed up one page, if possible. If the scroll is successful the routine attempts to move to a field on the new page that corresponds to the position of the current field in the current page.

If it is not possible to move up a page the routine returns SED\_STUCK.

If the sed is in active mode, **sed\_PageUp** calls the **fexit** function for the current field. If **fexit** returns FALSE, this routine does nothing and returns SED\_INVALID. If there are no fields on the new page, the current field remains the same (though it is not visible). Otherwise, the routine changes the current field to the new field. Before it enters the new field, it calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_PageUp** does not call the **fexit** and **fenter** functions.

### Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it is not possible to move up a page.

### See Also

**sed\_PageDown**, **sed\_PageLeft**, **sed\_PageRight**

### Example

```
switch (scancode) {  
case PGUP:  
    sed_PageUp(sed);  
    return(TRUE);  
case PGDN:  
    sed_PageDown(sed);  
    return(TRUE);  
/* ... */
```

### Synopsis

```
void sed_Pop(sed);  
    sed_type sed;           the sed
```

### Description

This routines removes a sed's image from the display by firing its window. The window manager repaints the windows that were obscured by the sed.

The sed and its window are not destroyed. To replace the sed onto the display call **sed\_Repaint**. To destroy the sed and its window call **sed\_Close**.

Refer to the chapter "Seds" in the *C-scape Manual* for more information.

### Return Value

There is no return value.

### See Also

**sed\_Close, sed\_Repaint**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
sed_Repaint(sed);  
sed_Go(sed);  
  
sed_Pop(sed);  
/* ... */
```

### Synopsis

```
void sed_ProtectField(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### Description

This routine “protects” a field. The user cannot enter a protected field. The field movement commands, such as **sed\_DecField** or **sed\_IncField**, skip over protected fields.

Protected fields are useful for fields you do not want the user to edit.

Fields can be unprotected with the **sed\_UnProtectField** command.

### Return Value

There is no return value.

### See Also

**menu\_Printf**, **sed\_UnProtectField**, **sed\_IsProtectedField**

### Note

This routine is implemented as a macro.

### Example

```
sed_ProtectField(sed, 0);
```

### Synopsis

```
char sed_PullLeft(sed);  
    sed_type sed;           the sed
```

### Description

This routine deletes the current character in the current field, and pulls the characters to the left of the deleted character over to fill the gap.

In the diagram below, the field is enclosed in a box and the underscore represents the current cursor position.

```
sed_PullLeft(sed);
```

has the following effect:

before:

helxlo mom

after:

hello mom

----> returns('x')

### Return Value

The character that was deleted.

### See Also

**sed\_Overwrite, sed\_PullRight, sed\_PushLeft, sed\_PushRight**

### Example

```
sed_PullLeft(sed);
```

### Synopsis

```
char sed_PullRight(sed);  
    sed_type sed;          the sed
```

### Description

This routine deletes the current character in the current field, and pulls the characters to the right of the deleted character over to fill the gap.

In the diagram below, the field is enclosed in a box and the underscore represents the current cursor position.

```
sed_PullRight(sed);
```

has the following effect:

before:	<table border="1"><tr><td>he<u>x</u>llo mom</td></tr></table>	he <u>x</u> llo mom	
he <u>x</u> llo mom			
after:	<table border="1"><tr><td>he<u>l</u>lo mom</td></tr></table>	he <u>l</u> lo mom	----> returns ('x')
he <u>l</u> lo mom			

### Return Value

The character that was deleted.

### See Also

**sed\_Overwrite, sed\_PullLeft, sed\_PushLeft, sed\_PushRight**

### Example

```
/* string_fkey ... */  
scancode = kb_Read();  
  
switch(scancode) {  
case BACKSPACE:  
    if (sed_DecChar(sed))  
        sed_PullRight(sed);  
    break;  
case DEL:  
    sed_PullRight(sed);  
    break;
```

### Synopsis

```
char sed_PushLeft(sed, c);  
  
    sed_type sed;           the sed  
    int c;                  the character to insert
```

### Description

This routine inserts the given character into the current field's record, and displaces characters to the left to accommodate the inserted character.

In the diagram below, the field is enclosed in a box and the underscore represents the current cursor position.

```
sed_PushLeft(sed, 'e');
```

has the following effect:

before:

q <u>h</u> llo mom
--------------------

after:

h <u>e</u> llo mom
--------------------

 ----> returns ('q')

### Return Value

Returns the character that falls off the left edge. If no character falls off, '\0' is returned.

### See Also

**sed\_Overwrite, sed\_PullLeft, sed\_PullRight, sed\_PushRight**

### Example

```
sed_PushLeft(sed, key);
```

## sed\_PushRight

Insert a character, pushing right

---

### Synopsis

```
char sed_PushRight(sed, c);  
  
    sed_type sed;          the sed  
    int c;                 the character to insert
```

### Description

This routine inserts the given character into the current field's record, and displaces characters to the right to accommodate the inserted character.

In the diagram below, the field is enclosed in a box and the underscore represents the current cursor position.

```
sed_PushRight(sed, 'e');
```

has the following effect:

```
before:  hllo mom  q  
  
after:   hello mom  ----> returns ('q')
```

### Return Value

Returns the character that fell off the right edge. If the cursor is past the end of the string, '\0' is returned.

### See Also

**sed\_Overwrite, sed\_PullLeft, sed\_PullRight, sed\_PushLeft**

### Example

```
if (isprint(key)) {          /* from string_fkey */  
    if (kb_Insert()) {  
        sed_PushRight(sed, key);  
    }  
    else {  
        sed_Overwrite(sed, key);  
    }  
    sed_IncChar(sed);  
}
```

### Synopsis

```
void sed_RedirectPrompt(sed1, sed2);  
    sed_type sed1;           target sed  
    sed_type sed2;           destination sed
```

### Description

This routine redirects all *sed1*'s border prompts to *sed2*'s border. This can be useful when you are using embedded sed's. You must ensure that both *sed1* and *sed2* are open and valid. You must not close *sed2* before *sed1*. *sed1* need not have a border for its prompts to be redirected; for this to be effective, however, *sed2* must have a border that supports the display of a prompt strings.

### Return Value

There is no return value.

### See Also

**sed\_SetBorder**

### Note

This routine is implemented as a macro.

### Example

```
sed_type sed1, sed2;  
  
/* ... */  
  
sed_RedirectPrompt(sed1, sed2);
```



### Synopsis

```
void sed_Repaint(sed);  
  
    sed_type sed;           the sed
```

### Description

This routine repaints the entire sed. If there is a border attached to the sed, **sed\_Repaint** paints that also. If there are any dependent bobs attached to the sed's fields, **sed\_Repaint** paints that also. Before painting the fields, **sed\_Repaint** calls the **sender** functions for each field in order to convert the user data to a displayable format.

The sed's window is hired if it is not currently employed.

You should call this routine before calling **sed\_Go**.

### Return Value

There is no return value.

### See Also

**sed\_RepaintField**, **sed\_Update**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
sed_Repaint(sed);  
sed_Go(sed);  
  
sed_Pop(sed);  
/* ... */
```

### Synopsis

```
void sed_RepaintBorder(sed);  
    sed_type sed;          the sed
```

### Description

This routine repaints the border attached to a sed. If there is no border attached to the sed this routine does nothing.

### Return Value

There is no return value.

### See Also

**sed\_Repaint**, **sed\_SetBorder**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
sed_SetBorderColor(sed, 0x70);  
sed_RepaintBorder(sed);  
/* ... */
```

### Synopsis

```
void sed_RepaintField(sed, fieldno);  
    sed_type sed;        the sed  
    int fieldno;         the field number
```

### Description

This routine repaints field *fieldno*. It calls the **senter** function for the field before painting the field.

### Return Value

There is no return value.

### See Also

**sed\_UpdateField**

### Example

```
/* x is the variable for field 2 */  
  
x = 7;  
  
sed_RepaintField(sed, 2);
```

### Synopsis

```
void sed_RepaintFields(sed);  
    sed_type sed;          the sed
```

### Description

This routine repaints all the sed's fields. It calls the **sender** function for each field before painting the fields. It only paints the fields, it does not paint the text buffer.

### Return Value

There is no return value.

### See Also

**sed\_UpdateFields**

### Note

This routine is implemented as a macro.

### Example

```
sed_RepaintFields(sed);
```

## sed\_RepaintRows

Refresh rows of a sed on the display

---

### Synopsis

```
void sed_RepaintRows(sed, start, end);  
  
    sed_type    sed;           sed  
    int         start;        start row  
    int         end;          end row
```

### Description

This routine refreshes the display image of a block of rows, from *start* through *end*, in *sed*. *start* and *end* are relative to the sed—**sed\_RepaintRows** takes the sed offsets into account, automatically.

Unlike **sed\_Repaint**, this function does not call the field **sender** functions. The contents of the field records are undisturbed.

### Return Value

There is no return value.

**Synopsis**

```
void sed_RewindTB(sed);  
    sed_type sed;          the sed
```

**Description**

This routine move the text cursor to the start of the text buffer without updating the display. It is usually called before reading the contents of the text buffer with **sed\_GetTB**.

Refer to the chapter “Text Editing” in the *C-scape Manual* for more information.

**Return Value**

There is no return value.

**See Also**

**sed\_GetTB**, **sed\_SetTB**

**Example**

```
char buffer[BUFLen];  
unsigned count;  
  
/* Rewind to start of the text buffer */  
sed_RewindTB(sed);  
  
/* Copy the text buffer into buffer*/  
count = sed_GetTB(sed, buffer, BUFLen, TED_HARD);  
  
/* ... */
```

### Synopsis

```
int sed_RightField(sed);  
    sed_type sed;          the sed
```

### Description

This routine takes the field to the right of the current field and makes it the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_RightField** does the following. If the sed is in active mode it calls the **fexit** function for the current field. If **fexit** returns FALSE, it does nothing and returns SED\_INVALID. If the current field is the right-most unprotected field the routine does nothing and returns SED\_STUCK. Otherwise, the field to the right becomes the new current field. Before it enters the new field, **sed\_RightField** calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_RightField** does not call the **fexit** and **fenter** functions.

You must not call this routine in either a field's **fenter** or **fexit** routines.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. A “p” following the number represents a protected field. Calling **sed\_RightField** from field 4 would make field 5 the new current field. Calling **sed\_RightField** from field 0 would make field 2 the new current field (field 1 is protected).

0	1p	2
3	4	5
6	7	8

## Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it could not find an unprotected new field to which it can move.

## See Also

**sed\_UpField, sed\_LeftField, sed\_DownField, sed\_ProtectField**

## Note

This routine is implemented as a macro.

## Example

```
switch (scancode) {
/* ... */
    case UP:
        sed_UpField(sed);
        return(TRUE);
    case DOWN:
        sed_DownField(sed);
        return(TRUE);
    case LEFT:
        sed_LeftField(sed);
        return(TRUE);
    case RIGHT:
        sed_RightField(sed);
        return(TRUE);
/* ... */
```



### Synopsis

```
int sed_ScrollDown(sed, lines);

    sed_type sed;           the sed
    int lines;              number of lines to scroll
```

### Description

This routine scrolls the sed down the specified number of lines, if possible. If the current field scrolls past the visible region, the routine tries to find a new, visible field to which it can move.

If the sed is in active mode, **sed\_ScrollDown** calls the **fexit** function for the current field. If **fexit** returns FALSE, this routine does nothing and returns SED\_INVALID. If there are no fields to move to, the current field remains the same (though it is not visible). Otherwise, the routine changes the current field to the new field. Before it enters the new field, it calls the **fenter** function for that field. If it is not possible to scroll, the routine returns SED\_STUCK.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_ScrollDown** does not call the **fexit** and **fenter** functions.

If there are no fields in a sed the scrolling functions simply scroll the text buffer.

### Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it is not possible to scroll.

### See Also

**sed\_ScrollLeft**, **sed\_ScrollRight**, **sed\_ScrollUp**

### Example

```
case HOME:
    sed_ScrollUp(sed, sed_GetYoffset(sed));
    break;
case END:
    sed_ScrollDown(sed, sed_GetMenuHeight(sed));
    break;
```

### Synopsis

```
int sed_ScrollLeft(sed, cols);  
    sed_type sed;           the sed  
    int cols;               number of columns to scroll
```

### Description

This routine scrolls the sed to the left the specified number of columns, if possible. If the current field scrolls past the visible region, the routine tries to find a new, visible field to which it can move.

If the sed is in active mode, **sed\_ScrollLeft** calls the **fexit** function for the current field. If **fexit** returns FALSE, this routine does nothing and returns SED\_INVALID. If there are no fields to move to, the current field remains the same (though it is not visible). Otherwise, the routine changes the current field to the new field. Before it enters the new field, it calls the **fenter** function for that field. If it is not possible to scroll, the routine returns SED\_STUCK.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_ScrollLeft** does not call the **fexit** and **fenter** functions.

If there are no fields in a sed the scrolling functions simply scroll the text buffer.

### Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it is not possible to scroll.

### See Also

**sed\_ScrollDown**, **sed\_ScrollRight**, **sed\_ScrollUp**

### Example

```
case LEFT:  
    sed_ScrollLeft(sed, 5);  
    break;  
case RIGHT:  
    sed_ScrollRight(sed, 5);  
    break;
```

### Synopsis

```
int sed_ScrollRight(sed, cols);  
    sed_type sed;           the sed  
    int cols;               number of columns to scroll
```

### Description

This routine scrolls the sed to the right the specified number of columns, if possible. If the current field scrolls past the visible region, the routine tries to find a new, visible field to which it can move.

If the sed is in active mode, **sed\_ScrollRight** calls the **fexit** function for the current field. If **fexit** returns FALSE, this routine does nothing and returns SED\_INVALID. If there are no fields to move to, the current field remains the same (though it is not visible). Otherwise, the routine changes the current field to the new field. Before it enters the new field, it calls the **fenter** function for that field. If it is not possible to scroll, the routine returns SED\_STUCK.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_ScrollRight** does not call the **fexit** and **fenter** functions.

If there are no fields in a sed the scrolling functions simply scroll the text buffer.

### Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it is not possible to scroll.

### See Also

**sed\_ScrollDown**, **sed\_ScrollLeft**, **sed\_ScrollUp**

### Example

```
case CTRL_LEFT:  
    sed_ScrollLeft(sed, sed_GetXoffset(sed));  
    break;  
case CTRL_RIGHT:  
    sed_ScrollRight(sed, sed_GetMenuWidth(sed));  
    break;
```

### Synopsis

```
int sed_ScrollUp(sed, lines);  
    sed_type sed;           the sed  
    int lines;              number of lines to scroll
```

### Description

This routine scrolls the sed up the specified number of lines, if possible. If the current field scrolls past the visible region, the routine tries to find a new, visible field to which it can move.

If the sed is in active mode, **sed\_ScrollUp** calls the **fexit** function for the current field. If **fexit** returns FALSE, this routine does nothing and returns SED\_INVALID. If there are no fields to move to, the current field remains the same (though it is not visible). Otherwise, the routine changes the current field to the new field. Before it enters the new field, it calls the **fenter** function for that field. If it is not possible to scroll, the routine returns SED\_STUCK.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_ScrollUp** does not call the **fexit** and **fenter** functions.

If there are no fields in a sed the scrolling functions simply scroll the text buffer.

### Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it is not possible to scroll.

### See Also

**sed\_ScrollDown**, **sed\_ScrollLeft**, **sed\_ScrollRight**

### Example

```
case HOME:  
    sed_ScrollUp(sed, sed_GetYoffset(sed));  
    break;  
case END:  
    sed_ScrollDown(sed, sed_GetMenuHeight(sed));  
    break;
```

### Synopsis

```
int sed_SearchMerge(sed, c);  
    sed_type sed;           the sed  
    char c;                 the search character
```

### Description

This routine finds the next field whose merge begins with the given character. It ignores leading spaces and case as well as protected fields and fields with writeable positions.

This routine is useful for implementing menus with first letter searching.

### Return Value

Returns the field number of the field if the search is successful; otherwise returns -1.

### Example

```
void menu_fkey(sed)  
    sed_type sed;  
{  
    int scancode, letter, choice;    /* from menu_fkey (fnmenu.c) */  
  
    scancode = kb_Read();  
  
    switch(scancode) {  
        /* ... */  
  
    default:  
        /* do first letter search */  
        if isprint(letter = ascii(scancode)) {  
            if ((choice = sed_SearchMerge(sed, (char) letter)) != -1){  
                sed_GotoField(sed, choice);  
            }  
        }  
        break;  
    }  
}
```

### Synopsis

```
void sed_SetActive(sed, mode);  
  
    sed_type sed;           the sed  
    boolean mode;          the active mode
```

### Description

This routine sets the active flag of the sed. The active flag indicates the active status of the sed.

### Return Value

There is no return value.

### See Also

`sed_GetActive`, `sed_Go`, `sed_IsActive`

### Note

This routine is implemented as a macro.

### Example

```
sed_SetActive(sed, TRUE);
```

### Synopsis

```
void sed_SetAux(sed, fun);
```

sed_type sed;	sed that gets auxiliary
aux_fptr fun;	auxiliary function

### Description

This routine attaches the auxiliary function *fun* to the given sed. If *fun* is NULL the sed will have no auxiliary function.

The auxiliary function is intended to provide the programmer with additional control over the operations of a sed without having to modify field functions. It allows processing before and after the field's **senter** function, **sexit** function, **fenter** function, and **fexit** function.

C-scape sends the following messages to auxiliary functions automatically:

SED_PRESENTER	called by a <b>Repaint</b> function immediately before the sed's <b>senter</b> functions are called. The auxiliary function's return value is ignored when it is sent this message.
SED_POSTSENDER	called by a <b>Repaint</b> function immediately after the sed's <b>senter</b> functions are called. The auxiliary function's return value is ignored when it is sent this message.
SED_PRESEXIT	called by <b>sed_Go</b> immediately before the sed's <b>sexit</b> functions are called. You can use this message to provide validation for an entire sed. If the auxiliary function returns FALSE in response to this message, the user will not be able to leave the sed.
SED_POSTSEXIT	called by <b>sed_Go</b> immediately after the sed's <b>sexit</b> functions are called. The auxiliary function's return value is ignored when it is sent this message.
SED_PREFENTER	called immediately before a field's <b>fenter</b> function is called. The auxiliary function's return value is ignored when it is sent this message.

- SED\_POSTFENTER** called immediately after a field's **fenter** function is called. The auxiliary function's return value is ignored when it is sent this message.
- SED\_PREFEXIT** called immediately before a field's **fexit** function is called. This message can provide extra field validation; if the auxiliary function returns **FALSE** when it receives this message, it is functionally equivalent to the **fexit** function returning **FALSE**.
- SED\_POSTFEXIT** called immediately after a field's **fexit** function is called (if the **fexit** function was successful). This message can provide extra field validation; if the auxiliary function returns **FALSE** when it receives this message, it is functionally equivalent to the **fexit** function returning **FALSE**.

The standard C-scape **SED\_** messages do not use *indata* or *outdata* (they are **NULL**). There are also auxiliary messages at the window and object levels; these are documented in the *OWL Manual* and *OWL Function Reference*.

C-scape provides one standard auxiliary function, called **aux\_Top**. It raises its sed above all other windows on the display when it becomes current by calling **sed\_Top** upon receiving a **WINA\_STARTGO** message.

You can use **sed\_DoAux** to send your own custom messages to an auxiliary function. Your message values should be unique and greater than **SED\_LASTMSG**.

All auxiliary functions must have the same form:



```

int aux_Sample(sed, msg, indata, outdata)
    sed_type sed;
    int msg;
    VOID *indata;
    VOID *outdata;
{
    switch(msg) {
    case SED_PRESENTER:
        /* ... */

    }

    return(1);
}

```

## **Return Value**

There is no return value.

## **Note**

This routine is implemented as a macro.

## **See Also**

**sed\_DoAux**

## Example

```
/* use macro to prototype auxiliary function */
aux_func(aux_FirstField);

main()
{
    /* ... */

    sed_SetAux(sed, aux_FirstField);

    /* ... */
}

int aux_FirstField(sed, msg, indata, outdata)
    sed_type sed;
    int msg;
    VOID *indata;
    VOID *outdata;
/*
    This auxiliary function makes sure that a sed's current
    field is always reset to the first field
    when the sed is painted.
*/
{
    int fldno;

    switch(msg) {
    case SED_PRESENTER:
        sed_GotoFirstField(sed);
        break;
    }

    return(TRUE);
}
```

### Synopsis

```
int sed_SetBaton(sed, baton);  
  
    sed_type sed;           the sed  
    int baton;              baton value
```

### Description

This routine sets the value of the baton. The baton passes state information between the functions in the field function structure (**fenter**, **fexit**, **fkey**, **senter**, and **sexit** functions) and is used by **sed\_Go** as a return value.

For some typical uses of the baton, see **sed\_GetBaton**.

The baton is initialized to -1 when the sed is created.

### Return Value

There is no return value.

### See Also

**sed\_GetBaton**, **sed\_Go**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
switch (scancode) {  
case ESC:  
    sed_SetBaton(sed, 0);  
    sed_ToggleExit(sed);  
    break;  
/* ... */
```

### Synopsis

```
boolean sed_SetBorder(sed, border_func);  
    sed_type sed;           the sed  
    bd_fpnr border_func;    the border function
```

### Description

This routine creates a border from the given border function and sends an “open” message to it.

Once a border has been attached to the sed, the other sed routines automatically incorporate it into their actions. The display routines (**sed\_Repaint**, **sed\_Update**) tell the border to paint itself. The destructor routines (**sed\_Close**, **sed\_SetBorder**) tell the border to destroyed itself. The size and positioning functions compensate for the size of the border when doing their calculations.

If a sed already has a border, **sed\_SetBorder** destroys the old border and replaces it with the new border.

C-scape’s standard border function is **bd\_cua**. To use this border in graphics mode, you must **#include** the file *ogldecl.h* and call the routines **ogl\_Init** and **bdcua\_InitGraphics** prior to calling **sed\_SetBorder**. To use it in text mode, call **bdcua\_InitText** prior to calling **sed\_SetBorder**. You can also call the routine **bdcua\_InitCombo**, if you want to use either text or graphics mode. For more information on **bd\_cua**, refer to the *OWL Manual* and the *OWL Function Reference*.

### Return Value

Returns TRUE if successful. If unable to create the border it returns FALSE.

### See Also

**bdcua\_InitCombo**, **bdcua\_InitGraphics**, **bdcua\_InitText**, **ogl\_Init**, **sed\_SetBorderFeature**, **sed\_SetBorderTitle**

### Note

This routine is implemented as a macro.

## Example

```
/* ... */  
  
sed = sed_Open(menu);  
  
sed_SetBorder(sed, bd_cua);  
sed_SetBorderTitle(sed, "My data screen");  
  
sed_Repaint(sed);  
/* ... */
```

### Synopsis

```
void sed_SetBorderColor(sed, color);  
    sed_type sed;           the sed  
    byte color;            the color
```

### Description

This routine sets the color of a border (other than **bd\_cua** for graphics) attached to the sed to *color*. If there is no border then the routine does nothing.

### Return Value

There is no return value.

### See Also

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
bdcua_InitText();  
  
/* ... */  
  
sed = sed_Open(menu);  
  
sed_SetBorder(sed, bd_cua);  
sed_SetBorderTitle(sed, "My data screen");  
sed_SetBorderColor(sed, 0xf);  
  
sed_Repaint(sed);  
/* ... */
```

### Synopsis

```
void sed_SetBorderFeature(sed, feature);  
    sed_type sed;           sed whose border features to set  
    unsigned int feature;    the feature bit mask
```

### Description

This routine sets the features of the given sed's border as specified by *feature*. *feature* is a bit mask formed by the bitwise ORing (the “|” operator in C) of border feature values. Providing zero for *feature* turns off all features. Consult the *OWL Manual* for a complete list of border features.

### Return Value

There is no return value.

### See Also

**sed\_SetBorder**

### Note

This routine is implemented as a macro.

### Example

```
sed_SetBorder(sed, bd_cua);  
sed_SetBorderFeature(sed, BD_MOVE | BD_RESIZE | BD_OUTLINE);
```

### Synopsis

```
int sed_SetBorderTitle(sed, title);  
    sed_type sed;           the sed  
    char *title;           the title string
```

### Description

This routine sets the title of a titled border to the string pointed to by *title*. This routine does nothing if there is no border attached to the sed or if the attached border does not support titles.

### Return Value

Returns TRUE if successful, FALSE otherwise.

### See Also

**sed\_BorderPrompt**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
sed = sed_Open(menu);  
  
sed_SetBorder(sed, bd_cua);  
sed_SetBorderTitle(sed, "My data screen");  
  
sed_Repaint(sed);  
/* ... */
```



### Synopsis

```
void sed_SetColors(sed, regular, background, highlight);  
    sed_type sed;           the sed  
    byte regular;           attributes  
    byte background;  
    byte highlight;
```

### Description

This routine sets the colors of the sed. It takes a sed object and three attribute values. *regular* is the regular color of fields. *highlight* is the color of highlighted fields. *background* sets the background color of the sed.

Marking a field overrides the *regular* and *highlight* attributes (see **sed\_MarkField**). The background color affects all the text defined in color 0 by **menu\_Printf** and is the initial color of a border.

You can change the displayed colors of a sed by adjusting values in the attribute map (sed **disp\_SetMapEntry**).

### Return Value

There is no return value.

### See Also

**sed\_GetBorderColor**, **sed\_GetColors**, **sed\_MarkField**, **disp\_SetMapEntry**

### Example

```
/* ... */  
  
sed = sed_Open(menu);  
  
sed_SetColors(sed, 0x70, 0x70, 0x07);  
  
sed_Repaint(sed);  
/* ... */
```

### Synopsis

```
void sed_SetCurrRecord(sed, string);  
    sed_type sed;           the sed  
    char *string;          the new record string
```

### Description

This routine sets the contents of the current field's record equal to *string*. It does not paint the field.

### Return Value

There is no return value.

### See Also

`sed_GetCurrRecord`

### Note

This routine is implemented as a macro.

### Example

```
/* ... clear the current field */  
  
sed_SetCurrRecord(sed, "");  
sed_UpdateCurrField(sed);    /* update the display */  
  
/* ... */
```

### Synopsis

```
void sed_SetCursorType(sed, ctype);  
    sed_type sed;           the sed  
    unsigned int ctype;     the cursor type
```

### Description

This routine sets the size of the sed's cursor. The default cursor type is `CURSOR_NORMAL`.

The cursor type is one of the following values:

<code>CURSOR_NORMAL</code>	The standard cursor.
<code>CURSOR_NONE</code>	An invisible cursor.
<code>CURSOR_BLOCK</code>	A full-sized cursor.
<code>CURSOR_DASH</code>	A thin cursor located in the middle of the character location.
<code>CURSOR_HALF</code>	A cursor filling half of the character location.
<code>CURSOR_THIN</code>	A thin cursor.

If the sed is inactive, **sed\_SetCursorType** sets the sed's cursor type to *ctype*. If the routine is called from within a field function the cursor type is changed until control leaves the current field. At that time, **sed\_Go** changes the cursor back to the type it was when the field was entered.

Note that cursor sizes are hardware dependent. Some cursor sizes may not be available for certain systems.

### Return Value

There is no return value.

### See Also

**sed\_GetCursorType**

## Example

```
/* ... */
sed = sed_Open(menu);

/* Turn off cursor in sed */
sed_SetCursorType(sed, CURSOR_NONE);

sed_Repaint(sed);
sed_Go(sed);

/* ... */
```

## sed\_SetData

Set the sed's generic data pointer

---

### Synopsis

```
void sed_SetData(sed, data);  
    sed_type sed;           the sed  
    VOID *data;            pointer to data
```

### Description

This routine sets the sed's generic data pointer. The generic data pointer is a VOID \* pointer that you can use for attaching program-specific data to a sed.

Some typical uses for the generic data pointer are:

- (1) as a place to store an array of strings to be displayed as messages, and
- (2) as a place to store the last value of the field so that it can be recalled in case the user makes a mistake.

### Return Value

There is no return value.

### See Also

#### sed\_GetData

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
sed = sed_Open(menu);  
  
sed_SetData(sed, (VOID *) &my_struct);  
  
/* ... */
```

### Synopsis

```
void sed_SetExit(sed, state);  
    sed_type sed;           the sed  
    boolean state;         the new exit state
```

### Description

This routine sets the exit state of the sed.

### Note

This routine is implemented as a macro.

### Return Value

There is no return value.

### See Also

**sed\_GetExit, sed\_ToggleExit**

### Example

```
sed_SetExit(sed, TRUE);
```

## sed\_SetExplode

Attach an explode function to the sed

---

### Synopsis

```
void sed_SetExplode(sed, explode);  
    sed_type sed;           the sed  
    exp_fptr explode;       the explode function
```

### Description

This routine attaches an explode function to the sed. The window manager calls the explode function when the sed is initially painted to the display. The explode function paints images to the display before the sed is painted and is used to create “special effects” when creating windows.

Some standard explode functions are **exp\_std** and **exp\_BeamMeUp**.

### Return Value

There is no return value.

### Note

This routine is implemented as a macro.

### Example

```
sed_SetExplode(sed, exp_std);  
sed_Repaint(sed);
```

### Synopsis

```
boolean sed_SetFieldBob(sed, fieldno, bob);  
  
    sed_type sed;           the sed  
    int fieldno;           the field number  
    bob_type bob;         the bob object
```

### Description

This routine attaches a bob object to a field. If the field already has a bob, the old bob is detached but not closed. Be sure to get a handle to the old bob before calling this function so you can close or reattach it.

### Return Value

Returns TRUE is successful; otherwise, it returns FALSE.

### See Also

**menu\_Printf, sed\_CreateBob, sed\_GetFieldBob**

### Note

This routine is implemented as a macro.

### Example

```
    bob = sed_CreateBob(ised, BOB_DEPENDENT);  
  
    /* ... */  
  
    sed_SetFieldBob(osed, 3, bob);  
  
    /* ... */
```



### Synopsis

```
boolean sed_SetFieldData(sed, fieldno, datano, data);  
    sed_type sed;           the sed  
    int fieldno;           the field number  
    int datano;            the data number  
    VOID *data             pointer to data
```

### Description

This routine redirects one of the field's generic data pointers to point to *data*. Each field has at least one generic data pointer that the programmer can set. The data pointer is a VOID \* pointer. You can use the *datano* argument to indicate which data pointer you want to set (0 is the first data pointer).

A field normally has one data pointer but you can override this with the **menu\_Printf** “@fd” command when the field is defined. After a field had been defined, you cannot change the number of data pointers it has.

The standard field functions use the field generic data pointers in the following ways:

- (0) to store border prompt strings.
- (1) to store validation data.
- (2) to store formatting information.

### Return Value

Returns TRUE if successful. Returns FALSE if passed a bad data number.

### See Also

**menu\_Printf**, **sed\_GetFieldData**, **sed\_GetFieldDataCount**

### Note

This routine is implemented as a macro.

### Example

```
sed_SetFieldData(sed, 0, 1, (VOID *) "(0,100)");
```

### Synopsis

```
void sed_SetFieldName(sed, fieldno, name);

    sed_type sed;           the sed
    int fieldno;            the field number
    char *name;             the field name
```

### Description

This routine sets the field's name to the string to which *name* points. If the field already has a name, **sed\_SetFieldName** removes the old name. If *name* is equal to NULL then the field will have no name. You can determine a field's name with **sed\_GetFieldName**.

### Return Value

There is no return value.

### See Also

**sed\_GetFieldName**, **sed\_GotoNameField**, **sed\_GetNameNo**

### Note

This routine is implemented as a macro.

### Example

```
/* Give field 1 a name */
sed_SetFieldName(sed, 1, "Touchstone");
```

### Synopsis

```
void sed_SetFieldWidth(sed, fieldno, width);  
  
    sed_type sed;           the sed  
    int fieldno;           the field number  
    int width;             the field width
```

### Description

This routine sets the displayed width of field *fieldno* to *width*. The displayed width of a field is its actual width on the display. This width is equal to the field's merge length unless you use **menu\_Printf** or **sed\_SetFieldWidth** to explicitly set its displayed width to a different value.

A field automatically scrolls when its merge length exceeds its displayed width and the user attempts to move the cursor past one of the field's edges.

### Return Value

There is no return value.

### See Also

**menu\_Printf**, **sed\_GetFieldWidth**

### Example

```
/* Make field 3 narrower */  
sed_SetFieldWidth(sed, 3, 10);
```

### Synopsis

```
void sed_SetFuncs(sed, fieldno, funcs);  
    sed_type sed;           the sed  
    int fieldno;           the field number  
    field_funcs_ptr funcs;  the field function structure
```

### Description

This routine sets the field's function structure. *funcs* must be the address of a valid field function structure.

### Return Value

There is no return value.

### See Also

[sed\\_GetFuncs](#)

### Note

This routine is implemented as a macro.

### Example

```
sed_SetFuncs(sed, 0, &string_funcs);
```

### Synopsis

```
void sed_SetHeight(sed, height);  
    sed_type sed;           the sed  
    int height;             the new height
```

### Description

This routine sets the height of the sed. If the height is smaller than the number of rows in the menu, the sed will automatically scroll.

*height* must be greater than zero.

The sed is re-sized on the display if its window is currently employed.

### Return Value

There is no return value.

### See Also

**sed\_SetWidth**

### Example

```
/* ... */  
  
sed = sed_Open(menu);  
  
sed_SetHeight(sed, 15);  
  
/* ... */
```

**Synopsis**

```
void sed_SetLabel(sed, label);  
    sed_type sed;           the sed  
    int label;              the label
```

**Description**

This routine sets the value of the sed's label.

The label is an integer that gives a unique number to a sed. The label typically specifies a chapter number for the help system.

The label is initially set to 0.

**Return Value**

There is no return value.

**See Also**

**sed\_GetLabel**

**Note**

This routine is implemented as a macro.

**Example**

```
/* ... */  
  
sed = sed_Open(menu);  
  
sed_SetLabel(sed, 1);  
  
/* ... */
```

### Synopsis

```
void sed_SetMouse(sed, mhandler);  
    sed_type sed;           the sed to which to attach  
    mouhandler_fptr mhandler; the mouse handler
```

### Description

This routine attaches the mouse handler *mhandler* to the given sed.

These mouse handlers are currently implemented:

**winmou\_All** provides mouse support for all situations except with a framer menuing system.

**sedmou\_Framer** provides mouse support for the framer menuing system.

To give **winmou\_All** the particular behavior you want, use the routine **win\_SetMouseFeature**. To use the mouse to size, move, and scroll the window you must attach a border to the sed with enabled mouse support. Other borders can have mouse handling capabilities enabled by using **sed\_SetBorderFeature**.

### Return Value

There is no return value.

### See Also

**win\_SetMouseFeature**, **sed\_SetBorderFeature**, **sed\_SetBorder**

## Example

```
menu_type menu;
sed_type popsed;
int      ret;

/* make a popup fun window to place above sed 'sed' */
menu = menu_Open();
menu_Printf(menu, "\n This is a popup window\n\n");
menu_Printf(menu, "@f[ OK ]      ", NULL, &menu_funcs);
menu_Printf(menu, "@fd2[Cancel]", NULL, &menu_funcs, NULL, "0");

popsed = sed_Open(menu);
sed_SetHeight(popsed, 5);
sed_SetWidth(popsed, 25);

/* get the position of the original sed */
sed_GetPosition(sed, &row, &col);

/* offset the popup from the original window */
sed_SetPosition(popsed, row + 3, col + 4);

sed_SetBorder(popsed, bd_cua);

sed_SetBorderFeature(popsed, BD_MOVE | BD_RESIZE);

sed_SetShadow(popsed, 1);
sed_SetShadowAttr(popsed, 0x08);

/* attach a mouse handler to the sed */
sed_SetMouse(popsed, sedmou_GreedyClick);
sed_Repaint(popsed);

ret = sed_Go(popsed);

sed_Close(popsed);

/* ... */
```



## sed\_SetNameVar

Point a named field's variable to new space

---

### Synopsis

```
boolean sed_SetNameVar(sed, name, var);  
  
    sed_type sed;           the sed  
    char *name;             the name of the field  
    VOID *var;              the new variable space
```

### Description

This routine finds the field in *sed* named *name*. It points this field's variable to the new space *var*. If it can't find a field named *name* in *sed*, it returns FALSE; otherwise, it returns TRUE.

This function is often used to attach variable space to seds that are loaded from screen files with the SED\_NOALLOC flag.

### Return Value

If it can't find a field named *name* in *sed*, it returns FALSE, otherwise it returns TRUE.

### See Also

**sed\_SetNameVarValue**

## Example

```
sfile_type sfile;
sed_type sed;
char item[101];
int quant;

/* load and validate "my_sed" */
if ((sfile = sfile_Open("sfile.lnf", my_symbollist)) == NULL
    || (sed = sfile_LoadSed(sfile, "my_sed", SED_NOALLOC)) == NULL
    || sed_SetNameVar(sed, "item", (VOID *)item) == FALSE
    || sed_SetNameVar(sed, "quant", (VOID *)&quant) == FALSE) {

    opc_Prompt(NULL, NULL, NULL, "Error in loading 'my_sed'");
    if (sfile != NULL) {
        sfile_Close(sfile);
    }

    return;
}
```

## **sed\_SetNameVarValue**      Copy a value into a named field's variable

---

### **Synopsis**

```
boolean sed_SetNameVarValue(sed, name, var);  
  
    sed_type sed;           the sed  
    char *name;            the name of the field  
    VOID *var;             a pointer to the new value
```

### **Description**

This routine finds the field in *sed* named *name*. It copies the new value pointed to by *var* into this field's variable. If it can't find a field named *name* in *sed*, it returns FALSE; otherwise, it returns TRUE.

This function is often used to initialize variables in seds that are loaded from screen files with the SED\_ALLOC flag.

### **Return Value**

If it can't find a field named *name* in *sed*, it returns FALSE, otherwise it returns TRUE.

### **See Also**

**sed\_SetNameVar**

## Example

```
sfile_type sfile;
sed_type sed;
static char default_item[101] = "Spinal Tap - Shark Sandwich";
static int default_qty = 1000;

/* load, validate, and initialize "my_sed" */
if ((sfile = sfile_Open("sfile.lnf", my_symbollist)) == NULL
    || (sed = sfile_LoadSed(sfile, "my_sed", SED_ALLOC)) == NULL
    || sed_SetNameVarValue(sed, "item", (VOID *) &default_item) ==
FALSE
    || sed_SetNameVarValue(sed, "quant", (VOID *) &default_qty) ==
FALSE) {

    opc_Prompt(NULL, NULL, NULL, "Error in loading 'my_sed'");
    if (sfile != NULL) {
        sfile_Close(sfile);
    }

    return;
}
```

### Synopsis

```
void sed_SetNextWin(sed, nextwin);  
    sed_type sed;           the current sed  
    win_type nextwin;       the next window
```

### Description

Use this routine if you wish to pass control to another window (*sed*) without leaving **sed\_Go**.

If you call **sed\_SetNextWin** with a handle to the next window, when you leave your *sed* by calling **sed\_ToggleExit**, the window manager will pass control to the *nextwin* instead of returning from **sed\_Go**. If *nextwin* is `NULL` the window manager will not pass control to another window and **sed\_Go** will simply return.

Typically this function is used by mouse handlers to pass control between various windows when they are selected with the mouse. The next window is initially set to `NULL`.

Note that *nextwin* can be a *sed* because a *sed* is a type of window.

### Return Value

There is no return value.

### Note

This routine is implemented as a macro.

### See Also

**sed\_Go**

### Example

```
case TAB:  
    /* Jump to other window (sed2) */  
    sed_SetNextWin(sed, sed2);  
    sed_ToggleExit(sed);  
    break;
```

### Synopsis

```
void sed_SetPosition(sed, row, col);
```

sed_type sed;	the sed
int row, col;	the display position of the sed

### Description

This routine sets the position of the upper-left corner of the sed. If there is a border attached to the sed the position refers to the upper-left corner of the border.

The sed is moved to a new position on the display if its window is currently employed.

### Return Value

There is no return value.

### Note

This routine is implemented as a macro.

### See Also

#### sed\_GetPosition

### Example

```
/* ... */

sed = sed_Open(menu);

sed_SetBorder(sed, bd_cua);
sed_SetPosition(sed, 5, 20);

/* ... */
```

### Synopsis

```
void sed_SetRecord(sed, string, fieldno);  
    sed_type sed;           the sed  
    char *string;          the new record string  
    int fieldno;           the field number
```

### Description

This routine sets the contents of field *fieldno*'s record to the given string. It does not paint the field.

If *string* is longer than the field's record, **sed\_SetRecord** does not copy the extra characters; if *string* is shorter, it pads the record with '\0's (which are displayed as spaces).

### Return Value

There is no return value.

### See Also

**sed\_GetRecord**

### Note

This routine is implemented as a macro.

### Example

```
void string_senter(sed, fieldno)  
    sed_type sed;  
    int fieldno;  
/*  
    Copy the native string into the record string.  
*/  
{  
    sed_SetRecord(sed, (char *) sed_GetVar(sed, fieldno), fieldno);  
}
```

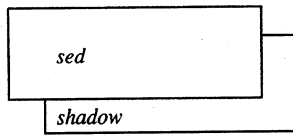
## Synopsis

```
void sed_SetShadow(sed, shd);  
  
    sed_type sed;           sed whose shadow to set  
    int shd;                size of shadow in characters
```

## Description

This routine sets the size of the given sed's shadow as specified by *shd*. Note that shadow size is specified in character units.

A shadow is an extension to the sed's border that paints a darkened strip along the bottom and right edges of the sed to give it a three dimensional appearance. The color of the strip is determined by the shadow attribute of the window on which the shadow falls.



## Return Value

There is no return value.

## Note

This routine is implemented as a macro.

## See Also

**sed\_SetShadowAttr**

## Example

```
sed_SetBorder(sed, bd_cua);  
sed_SetShadow(sed, 1);
```



### Synopsis

```
void sed_SetShadowAttr(sed, attr);  
    sed_type sed;           the sed  
    byte attr;              the shadow color attribute
```

### Description

This routine sets the sed's shadow attribute to the value given by *attr*.

The shadow attribute dictates how a shadow (from some other overlapping sed with a shadow border) will appear as it falls on this given sed. For example, if you want black shadows to appear across your sed use attribute 0x00. Use a different attribute, such as 0x08, to create shadows that show the contents of the sed faintly within the shadow.

### Return Value

There is no return value.

### Note

This routine is implemented as a macro.

### See Also

[sed\\_SetShadow](#)

### Example

```
/* ... */  
  
sed = sed_Open(menu);  
  
sed_SetShadow(sed, 1);  
sed_SetShadowAttr(sed, 0x08);  
  
/* ... */
```

### Synopsis

```
void sed_SetSpecial(sed, special);  
    sed_type sed;           the sed  
    spc_fptra special;      the special function
```

### Description

This routine attaches a special function to the sed. If special is NULL then the sed will have no special function.

You can execute the special function by calling **sed\_DoSpecial**.

The special function is called by the fkey function of the standard field functions. It is used to customize the operation of all the fields in a sed without having to modify any field functions.

Several special functions, such as **spc\_Abort** and **spc\_Embed**, are provided and you can create your own as needed.

### Return Value

There is no return value.

### See Also

**sed\_DoSpecial**, **spc\_func**

### Note

This routine is implemented as a macro.

## Example

```
/* use macro to prototype special function */
spc_func(my_spc_func);

main()
{
    /* ... */
    sed = sed_Open(menu);

    /* Make Escape key return SED_ABORT */
    sed_SetSpecial(sed, my_spc_func);
}
```

**Synopsis**

```
boolean sed_SetTB(sed, text, len);  
    sed_type sed;           the sed  
    char *text;             the text array  
    unsigned int len;       length of the text array
```

**Description**

This routine sets the contents of the sed's text buffer. It copies the first *len* characters from *text* into the text buffer starting at the current text cursor position and advances the cursor position the number of characters set. This makes it possible to call **sed\_SetTB** repeatedly to load large arrays of text.

You can clear the contents of the text buffer prior to calling this routine with **sed\_ClearTB**.

**Return Value**

Returns TRUE if successful, FALSE otherwise.

**See Also**

**sed\_GetTB, sed\_ClearTB**

**Example**

```
char buffer[BUFLen];  
  
/* clear the contents of the text buffer */  
sed_ClearTB(sed);  
  
/* copy buffer into the text buffer */  
sed_SetTB(sed, buffer, BUFLen);  
  
/* ... */
```

**Synopsis**

```
void sed_SetVar(sed, fieldno, var);  
    sed_type sed;           the sed  
    int fieldno;           the field number  
    VOID *var;             pointer to new field variable
```

**Description**

This routine sets the pointer to field *fieldno*'s variable to a pointer to a new variable. The field's variable pointer is initially set when it is defined by **menu\_Printf**. **sed\_SetVar** lets you make a field point to a new variable.

The field variable is a VOID \* pointer. Use a type cast to convert another type to a VOID \* pointer when necessary.

**Return Value**

There is no return value.

**See Also**

**menu\_Printf**, **sed\_GetVar**

**Note**

This routine is implemented as a macro.

**Example**

```
int val;  
  
sed_SetVar(sed, 0, (VOID *) &val);
```

### Synopsis

```
void sed_SetVarValue(sed, fieldno, value);  
  
    sed_type sed;           the sed  
    int fieldno;           the field number  
    VOID *value;           pointer to new field variable
```

### Description

This routine copies the value pointed to by *value* into field *fieldno*'s variable. It can be used to initialize a fields variable or to give a fields variable a new value.

The field variable is a VOID \* pointer. *value* must be a void pointer to the new value. Use a type cast to convert another type to a VOID \* pointer when necessary.

### Return Value

There is no return value.

### See Also

**sed\_GetVar, sed\_GetVarValue**

### Example

```
int dummy_zero = 0;  
  
sed_SetVarValue(sed, 0, (VOID *) &dummy_zero);  
sed_SetVarValue(sed, 1, (VOID *) &dummy_zero);
```

### Synopsis

```
void sed_SetWidth(sed, width);  
    sed_type sed;           the sed  
    int width;              the new width
```

### Description

This routine sets the width of the sed. If the width is smaller than the menu, you can use the scrolling functions to display all of it.

*width* must be greater than zero.

The sed is re-sized on the display if its window is currently employed.

### Return Value

There is no return value.

### See Also

#### sed\_SetHeight

### Example

```
sed = sed_Open(sed);  
  
sed_SetWidth(sed, 60);
```

### Synopsis

```
void sed_SwapFields(sed, field1, field2);  
    sed_type sed;           the sed  
    int field1;             the field number of the first field  
    int field2;             the field number of the second field
```

### Description

This routine swaps the field numbers of fields *field1* and *field2*. The field locations and other data do not change.

The sed is not repainted to show the fields in their new positions. Use either **sed\_Repaint** or **sed\_Update** for this purpose.

### Return Value

There is no return value.

### See Also

**sed\_MoveField**

### Note

This routine is implemented as a macro.

### Example

```
/* Swap field 1 and 22 */  
sed_SwapFields(sed, 1, 22);
```



### Synopsis

```
void sed_ToggleExit(sed);  
    sed_type sed;           the sed
```

### Description

This routine causes the sed to exit and return control to the calling function. The sed still checks the **fxit** function for permission to leave before it does so.

### Return Value

There is no return value.

### See Also

**sed\_GetExit, sed\_Go**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
switch (scancode) {  
case ESC:  
    sed_SetBaton(sed, 0);  
    sed_ToggleExit(sed);  
    return(TRUE);  
case ENTER:  
    /* try to go to the next field else exit */  
    if (sed_IncField(sed) == SED_STUCK) {  
        sed_SetBaton(sed, sed_GetFieldNo(sed)+1);  
        sed_ToggleExit(sed);  
    }  
    return(TRUE);  
}  
  
/* ... */
```

**Synopsis**

```
void sed_Top(sed);  
    sed_type sed;          the sed
```

**Description**

This routine brings a sed to the top of the window list. If the sed is behind any other windows it will be repainted in the foreground. This routine does not repaint anything if the sed's window is unemployed or if it is already on top.

**Return Value**

There is no return value.

**Note**

This routine is implemented as a macro.

**Example**

```
/* Make sure sed is visible... */  
sed_Top(sed);
```

### Synopsis

```
void sed_UnMarkField(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### Description

This routine “unmarks” a field previously marked by **sed\_MarkField**. You can mark the field again with **sed\_MarkField**.

The unmarked field, when repainted, will be drawn with the current sed field colors.

### Return Value

There is no return value.

### See Also

**sed\_MarkField**, **sed\_IsMarkedField**

### Note

This routine is implemented as a macro.

### Example

```
sed_UnMarkField(sed, 2);  
sed_UpdateField(sed, 2);
```

### Synopsis

```
void sed_UnProtectField(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### Description

This routine “unprotects” a field previously protected by **sed\_ProtectField** or **menu\_Printf**.

You can protect the field again with **sed\_ProtectField**.

### Return Value

There is no return value.

### See Also

**sed\_ProtectField**, **sed\_IsProtectedField**

### Note

This routine is implemented as a macro.

### Example

```
sed_UnProtectField(sed, 22);
```

## sed\_Update

Paint a sed without converting data

---

### Synopsis

```
void sed_Update(sed);  
    sed_type sed;           the sed
```

### Description

This routine draws a sed on the display without calling the **sender** functions for the fields.

### Return Value

There is no return value.

### See Also

**sed\_Repaint, sed\_UpdateFields**

### Note

This routine is implemented as a macro.

### Example

```
/* Clear the fields */  
for (i = 0; i < sed_GetFieldCount(sed); i++) {  
    sed_SetRecord(sed, "", i);  
}  
  
sed_Update(sed);
```

### Synopsis

```
void sed_UpdateCurrField(sed);  
    sed_type sed;          the sed
```

### Description

This routine paints the contents of the current field to the display without calling the **sender** function for the field.

### Return Value

There is no return value.

### See Also

**sed\_RepaintField**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
sed_SetCurrRecord(sed, "Empty");  
sed_UpdateCurrField(sed);  
  
/* ... */
```

## sed\_UpdateField

Paint a field without converting data

---

### Synopsis

```
void sed_UpdateField(sed, fieldno);  
    sed_type sed;           the sed  
    int fieldno;           the field number
```

### Description

This routine paints the contents of the specified field to the display without calling the **sender** function.

### Return Value

There is no return value.

### See Also

**sed\_RepaintField**

### Note

This routine is implemented as a macro.

### Example

```
/* ... */  
  
sed_SetRecord(sed, 0, "");  
sed_UpdateField(sed, 0);  
  
/* ... */
```

### Synopsis

```
void sed_UpdateFields(sed);  
    sed_type sed;          the sed
```

### Description

This routine paints the contents of all the visible fields to the display without calling the **sender** functions for the fields. **sed\_UpdateFields** paints only the fields, not the text buffer.

### Return Value

There is no return value.

### See Also

**sed\_RepaintFields**

### Note

This routine is implemented as a macro.

### Example

```
/* Fill fields with ... */  
for (i = 0; i < sed_GetFieldCount(sed); i++) {  
    sed_SetRecord(sed, "....", i);  
}  
  
sed_UpdateFields(sed);
```



## sed\_UpField

Move up a field

### Synopsis

```
int sed_UpField(sed);  
    sed_type sed;           the sed
```

### Description

This routine takes the field above the current field and makes it the new current field. If it is called within an active sed it highlights the new field and moves the cursor to the first writeable position (if there is one) of the new field.

In detail, **sed\_UpField** does the following. If the sed is in active mode, it calls the **fexit** function for the current field. If **fexit** returns FALSE, it does nothing and returns SED\_INVALID. If the current field is the top-most unprotected field the routine does nothing and returns SED\_STUCK. Otherwise, to the field above the current field becomes the new current field. Before it enters the new field, **sed\_UpField** calls the **fenter** function for that field.

If the sed is not in active mode (i.e., if the routine is called outside of **sed\_Go**), **sed\_UpField** does not call the **fexit** and **fenter** functions.

You must not call this routine in either a field's **fenter** or **fexit** routines.

In the diagram below, the boxes represent fields. The numbers within the boxes represent the numbering of the fields. A "p" following the number represents a protected field. Calling **sed\_UpField** from field 4 would make field 1 the new current field. Calling **sed\_UpField** from field 8 would make field 2 the new current field (field 5 is protected).

0	1	2
3	4	5p
6	7	8

## Return Value

Returns SED\_MOVED if successful, SED\_INVALID if the original field's **fexit** function failed, and SED\_STUCK if it could not find an unprotected new field to which it can move.

## See Also

**sed\_DownField, sed\_LeftField, sed\_RightField, sed\_ProtectField**

## Note

This routine is implemented as a macro.

## Example

```
switch (scancode) {
/* ... */
    case UP:
        sed_UpField(sed);
        return(TRUE);
    case DOWN:
        sed_DownField(sed);
        return(TRUE);
    case LEFT:
        sed_LeftField(sed);
        return(TRUE);
    case RIGHT:
        sed_RightField(sed);
        return(TRUE);
/* ... */
}
```

**Synopsis**

```
void sedwin_MouseInit();
```

**Description**

This routine initializes the request handler for the sed window class to enable support for mouse functions. The handler affects all sed's and all borders. It permits them to respond to mouse messages that pertain to scrolling, re-sizing, and dragging.

You only need to call this routine if you want to use the mouse with sed's. It is not necessary if you are not using the mouse—a null handler is used by default.

**Return Value**

There is no return value.

**See Also**

**hard\_InitMouse, sed\_SetMouse, sed\_SetBorder, sed\_SetBorderFeature**

**Example**

```
void main()
{
    /* Initialize device interface */
    disp_Init(def_ModeCurrent, NULL);

    /* Turn on the mouse */
    hard_InitMouse();

    /* Turn on sedwin mouse support */
    sedwin_MouseInit();
}
```

### Synopsis

```
#include "sfile.h"

sed_type sfile_LoadSed(sfile, sedname, mode);
    sfile_type sfile;           the screen file
    char *sedname;             the name of the sed
    int mode;                   the auto allocation flag
```

### Description

This routine loads the sed named by *sedname* from the screen file *sfile* and, dependent upon *mode*, may allocate space for the sed's field variables.

If SED\_ALLOC is passed as *mode* then this routine allocates whatever storage the sed's fields require (using the *varsize* element of each field's function structure) and points the field's variables to this storage. You can get at this storage with **sed\_GetVar**. If you name the fields in a screen file with Look & Feel, or have otherwise named them in **menu\_Printf** or with **sed\_SetFieldName**, you can then use **sed\_GetNameVar** to get a pointer to the field's variable.

The storage allocated by **sfile\_LoadSed** is released when you close the sed with **sed\_Close**.

If *mode* is equal to SED\_NOALLOC, **sfile\_LoadSed** does not allocate storage space. You must explicitly declare variables or otherwise allocate storage and then point the field variables to this storage with the **sed\_SetVar** routine. Note that if you do not use auto-allocation you must ensure that **every** field's variable points to valid storage space or else the field's senter and sexit functions will not function properly. For specialized seds such as sleds and teds, space is already allocated automatically, so SED\_NOALLOC should be specified.

Before you call **sfile\_LoadSed** you must have successfully opened a screen file with **sfile\_Open**. The symbol list that was passed to **sfile\_Open** must include entries for all the functions used in a screen file. If this symbol list is incomplete **sfile\_LoadSed** will not be able to match screen file symbols with the appropriate functions. Specifically, it will use **null\_funcs** for any field function symbol it cannot find and NULL for any other type of function symbol it cannot find. You should

try, however, not to include symbols for unnecessary functions. If you do, these routines will be linked into your program and add unnecessary code to your application. You can use Look & Feel to automatically generate a symbol table for a given screen file.

You must include *sfile.h* in order to use **sfile\_** routines.

## Return Value

Returns a handle to the sed it loaded or NULL if unable to load the sed.

## See Also

**sfile\_Open**, **sfile\_SaveSed**, **sfile\_Close**

## Example

```
fsyminit_struct my_list[] = {
{FSYM_CLASS},
/* don't need to declare seds--they're implicit! */

{FSYM_FIELDFUNCS},
{"datet_funcs",      FNULL, (VOID *)  &datet_funcs      },
{"slong_funcs",      FNULL, (VOID *)  &slong_funcs      },
{"string_funcs",     FNULL, (VOID *)  &string_funcs     },

{FSYM_BORDER},
{"bd_cua",           (VOID_FPTR) bd_cua      ,  NULL },

{FSYM_MOUSE},
{"winmou_All",       (VOID_FPTR) winmou_All  ,  NULL },

{FSYM_LISTEND}
};
```

```

void main()
{
    sed_type sed, sled;
    sfile_type sfile;

    /* Initialize the display */
    disp_Init(def_ModeText, NULL);

    sfile = sfile_Open("aliens.lnf", my_list);

    /* Load the sed, allocate space */
    sed = sfile_LoadSed(sfile, "saucer", SED_ALLOC);

    /* Load the sled, don't allocate space */
    sled = sfile_LoadSed(sfile, "galaxies_sled", SED_NOALLOC);

    if (sed != NULL) {
        sed_Repaint(sed);
        sed_Go(sed);
        /* pull the data out of the sed... */

        sed_Close(sed);
    }

    if (sled != NULL) {
        sed_Repaint(sled);
        sed_Go(sled);
        /* pull the data out of the sled... */

        sed_Close(sled);
    }

    sfile_Close(sfile);
    disp_Close();
}

```

**Synopsis**

```
#include "sfile.h"

boolean sfile_SaveSed(sfile, sed, sedname);

    sfile_type sfile;           screen file into which to save
    sed_type sed;              sed to save
    char *sedname;             name of the sed in the file
```

**Description**

This routine saves the given sed *sed* into the screen file *sfile*. *sedname* is the name by which the sed will be known when later loaded as a screen by **sfile\_LoadSed**. Note that the sed's name is a string and must be in quotation marks if used as a string constant (i.e., literal).

Before you call **sfile\_SaveSed** you must have successfully opened a screen file with **sfile\_Open**. The symbol list that was passed to **sfile\_Open** must include entries for all the functions used in the sed. If this symbol list is incomplete **sfile\_SaveSed** will not be able to match screen file symbols with the appropriate functions.

You must include *sfile.h* in order to use **sfile\_** routines.

**Return Value**

Returns TRUE if successful; FALSE, if not.

**See Also**

**sfile\_Open**, **sfile\_LoadSed**, **sfile\_Close**

## Example

```
fsyminit_struct my_list[] = {

    {FSYM_FIELDFUNCS },
    {"string_funcs", NULL, (VOID *) &string_funcs},

    {FSYM_USER },
    {FSYM_BORDER },
    {"bd_cua", (VOID_FPTR) bd_cua, NULL},

    {FSYM_LISTEND}
};

void main()
{
    menu_type menu;
    sed_type sed;
    sfile_type sfile;
    char name[22];

    name[0] = '\0';

    /* Initialize the display */
    disp_Init(def_ModeText, NULL);

    /* Open the sfile */
    sfile = sfile_Open("test.lnf", my_list);

    /* Create the sed */
    menu = menu_Open();
    menu_Printf(menu, "This is a test\n");
    menu_Printf(menu, "@f[#####]\n", name, &string_funcs);

    sed = sed_Open(menu);
    sed_SetBorder(sed, bd_cua);
    sed_SetBorderTitle(sed, "Test");
    sed_SetPosition(sed, 5, 10);

    /* Put the sed into the sfile */
    sfile_SaveSed(sfile, sed, "test");

    sfile_Close(sfile);
    disp_Close();
}
```



## sled\_DeleteRows

Destroy rows in a sled

---

### Synopsis

```
#include "sled.h"

void sled_DeleteRows(sled, row, count);

    sed_type sled;           the sled
    int row;                 row to delete
    int count;               number of rows to delete
```

### Description

This routine deletes *count* rows from the sled's column arrays at row *row*. The rows after the deleted rows are moved to close the gap created by the deletion.

### Return Value

There is no return value.

### See Also

**sled\_InsertRows**, **sled\_Repaint**

### Example

```
#include "sled.h"

/* ... */
case GREYPLUS:
    sled_InsertRows(sed, sled_GetRow(sed), 1);
    sled_Repaint(sed);
    return(TRUE);
case GREYMINUS:
    sled_DeleteRows(sed, sled_GetRow(sed), 1);
    sled_Repaint(sed);
    return(TRUE);
}
```

### Synopsis

```
#include "sled.h"

int sled_GetRow(sled);
    sled_type sled;

int sled_GetCol(sled);
    sled_type sled;
```

### Description

These routines determine the sled's current row or column number. The current row number corresponds to the element number within the column array that the current field's variable points to.

### Return Value

Returns the sled's current row or column number.

### See Also

**sled\_IsLastRow**

### Example

```
#include "sled.h"

/* ... */
case GREYPLUS:
    sled_InsertRows(sed, sled_GetRow(sed), 1);
    sled_Repaint(sed);
    return(TRUE);
case GREYMINUS:
    sled_DeleteRows(sed, sled_GetRow(sed), 1);
    sled_Repaint(sed);
    return(TRUE);
}
```

## sled\_GetColSize

Get the size of the sled's columns

---

### Synopsis

```
#include "sled.h"

int sled_GetColSize(sled);
    sed_type sled;          the sled
```

### Description

This routine observes the number of rows in the sled's column array(s). This number is always greater than or equal to 1.

### Return Value

Returns the number of rows in the sled's column array(s).

### See Also

**sled\_GetWidth, sled\_GetColVar**

### Note

This routine is implemented as a macro.

### Example

```
#include "sled.h"

/* Printf the data out */
for (row = 0; row < sled_GetColSize(sled); row++) {
    printf("%s ", (char *) sled_GetColVar(sled, 0, row));
    printf("%d\n", *((int *) sled_GetColVar(sled, 1, row)));
}
```

### Synopsis

```
#include "sled.h"

VOID *sled_GetColVar(sled, col, row);

    sed_type sled;           the sled
    int col;                 the col
    int row;                 the row
```

### Description

This routine returns a pointer to element *row* in column array *col*.

This routine returns a VOID \* pointer; you must cast this pointer to the appropriate data type.

### Return Value

Returns a pointer to an element in the sled's column array.

### See Also

#### sled\_SetColVar

### Example

```
#include "sled.h"

/* Printf the data out */
for (row = 0; row < sled_GetColSize(sled); row++) {
    printf("%s ", (char *) sled_GetColVar(sled, 0, row));
    printf("%d\n", *((int *) sled_GetColVar(sled, 1, row)));
}
```

### Synopsis

```
#include "sled.h"

int sled_GetWidth(sled);

    sed_type sled;          the sled
```

### Description

This routine observes the number of fields in each row in the sled's column array(s). This number is always greater than or equal to 1.

### Return Value

Returns the number of fields in each of the sled's rows.

### See Also

**sled\_GetColSize, sled\_GetColVar**

### Note

This routine is implemented as a macro.

### Example

```
#include "sled.h"

/* ... */

/* traverse the sled's row */
for (col = 0; col < sled_GetWidth(sled); col++) {

    /* do something to each field in this row of the sled */
}
```

### Synopsis

```
#include "sled.h"

boolean sled_InsertRows(sled, row, count);

    sed_type sled;           the sled
    int row;                 row to insert at
    int count;               number of rows to insert
```

### Description

This routine inserts *count* blanks rows into the sled's column arrays at row *row*.

### Return Value

Returns TRUE if it is able to insert the rows; otherwise, it returns FALSE..

### See Also

**sled\_DeleteRows, sled\_Repaint**

### Example

```
#include "sled.h"

/* ... */
case GREYPLUS:
    sled_InsertRows(sed, sled_GetRow(sed), 1);
    sled_Repaint(sed);
    return(TRUE);
case GREYMINUS:
    sled_DeleteRows(sed, sled_GetRow(sed), 1);
    sled_Repaint(sed);
    return(TRUE);
}
```

## sled\_IsLastRow

Check if the current row is the last

---

### Synopsis

```
#include "sled.h"

boolean sled_IsLastRow(sled);
    sed_type sled;          the sled
```

### Description

This routine tests if the current row in the sled corresponds to the last row in the sled's column arrays.

### Return Value

Returns TRUE if the current row in the sled corresponds to the last row in the sled's column arrays.

### See Also

**sled\_GetRow**, **sled\_Remap**

### Note

This routine is implemented as a macro.

### Example

```
#include "sled.h"

/* ... */
case DOWN:
    /* See if we've reached the end of the column array
       or the end of the sed */
    if (sled_IsLastRow(sed) || sed_DownField(sed) == SED_STUCK) {
        sed_SetBaton(sed, BOB_DOWN);
        sed_ToggleExit(sed);
    }
    return(TRUE);
```

## sled\_MarkCol

Create a scrolling list

### Synopsis

```
#include "sled.h"

void sled_GetMarkAttr(sed, regular, selected);
    sed_type  sed;
    byte *regular;
    byte *selected;

void sled_SetMarkAttr(sed, regular, selected);
    sed_type sed;
    byte regular;
    byte selected;

boolean sled_IsMarkedRow(sed, row);
    sed_type sed;
    int row;

boolean sled_IsMarkedCol(sed, col);
    sed_type sed;
    int col;

void sled_MarkCol(sed, col);
    sed_type sed;
    int col;

void sled_UnMarkCol(sed, col);
    sed_type sed;
    int col;

void sled_MarkRow(sed, row);
    sed_type sed;
    int row;

void sled_UnMarkRow(sed, row);
    sed_type sed;
    int row;
```

### Description



These functions mark rows and columns within a sled, set the marking colors, and test if a row or column is marked.

**sled\_GetMarkAttr** returns the current marking colors by placing them in *regular* and *selected*.

**sled\_SetMarkAttr** sets the sled's marking colors to the values *regular* and *selected*.

**sled\_IsMarkedRow** and **sled\_IsMarkedCol** check the marked status of a row or column.

**sled\_MarkCol**, **sled\_UnMarkCol**, **sled\_MarkRow**, and **sled\_UnMarkRow** mark or unmark a row or column in a sled.

### Synopsis

```
#include "sled.h"
```

```
sed_type sled_Open(menu, height, special);
```

menu_type menu;	the menu
int height;	height of the sled
spc_fptr special;	the special function

### Description

This routine creates a sed with *height* rows from a menu that defines the first row of the sed. **sled\_Open** also allocates a column array for each field in *menu*. These columns arrays grow as you add data to them. *special* is a sed special function bound to the sed to handle the scrolling list operation. You can destroy the sed and its column arrays with **sed\_Close**.

### Return Value

Returns a handle to a sed object or NULL if unable to create the sled.

### Note

This routine is implemented as a macro.

### See Also

**sed\_Close**

## Example

```
#include "sled.h"

{
    menu_type imenu, omenu;
    sed_type sled, osed;
    bob_type bob;

    /* Create one row of the sled */
    imenu = menu_Open();

    menu_Printf(imenu, "@f[#####]", NULL, &money_funcs);

    /* Create a sled 5 rows tall */
    sled = sled_Open(imenu, 5, spc_Sled);

    /* Create a bob from the sled */
    bob = sed_CreateBob(sled, BOB_DEPENDENT);

    /* Place the sled inside an outer sed */
    omenu = menu_Open();

    menu_Printf(menu, "@fb[]", NULL, &sled_funcs, bob);

    osed = sed_Open(omenu);

    sed_Repaint(osed);
    sed_Go(osed);

    /* ... Destroy all the objects */
    sed_Close(osed);
}
```

## **sled\_PageDown - sled\_PageUp** Sled page up and down routines

---

### **Synopsis**

```
#include "sled.h"

int sled_PageUp(sled);
    sed_type sled;        the sled

int sled_PageDown(sled);
    sed_type sled;        the sled
```

### **Description**

The two routines, **sled\_PageUp** and **sled\_PageDown**, facilitate scrolling within a sled that has fixed rows at its top. Each remaps a sled up or down by the number of scrollable rows in it.

### **Return Value**

Like **sled\_Remap**, these functions return **SED\_MOVED**, **SED\_STUCK**, or **SED\_INVALID**.

### Synopsis

```
#include "sledprot.h"

void sled_ProtectRow(sled, row);
    sled_type sled;
    int row;

void sled_ProtectCol(sled, col);
    sled_type sled;
    int col;

boolean sled_IsProtectedRow(sled, row);
    sled_type sled;
    int row;

boolean sled_IsProtectedCol(sled, col);
    sled_type sled;
    int col;
```

### Description

These functions allow you to protect a sled row or column or check the protected status of a sled row or column.

A user cannot enter a protected row or column. Protected rows and columns are useful for data you do not want the user to edit.

### Return Value

**sled\_IsProtectedRow** and **sled\_IsProtectedCol** return TRUE if the row or column is protected or FALSE otherwise.

### Synopsis

```
#include "sled.h"

boolean sled_Remap(sled, rows);
    sed_type sled;          the sled
    int rows;               number of rows to scroll sled
```

### Description

This routine points the sled's field variables to a new set of elements in the sled's column arrays. The new elements are *rows* rows away from the old elements in the column array. Negative numbers move the variables toward the top of the column array.

### Return Value

Returns TRUE if it is able to remap the sled's variables; otherwise, it returns FALSE.

### See Also

**sled\_GetRow**

## Example

```
#include "sled.h"

/* ... */
switch (sed_Go(sled)) {
case BOB_UP:
    if ( sled_Remap(sled, -1) ) {
        /* the field should appear to be newly entered */
        sed_DoFieldFenter(sed, sed_GetFieldNo(sed));
        sed_GoHome(sled);
    }
    else {
        sed_UpField(sed);
    }
    break;
case BOB_DOWN:
    if ( sled_Remap(sled, 1) ) {
        /* the field should appear to be newly entered */
        sed_DoFieldFenter(sed, sed_GetFieldNo(sed));
        sed_GoHome(sled);
    }
    else {
        sed_DownField(sed);
    }
    break;
```

### Synopsis

```
#include "sled.h"

void sled_Repaint(sled);

    sed_type sled;          the sled
```

### Description

This routine call the **sender** functions for the sled's fields and repaints the sled. If a field's variable points past the end of a column array, **sled\_Repaint** paints it blank.

### Return Value

There is no return value.

### See Also

**sled\_DeleteRows**, **sled\_InsertRows**

### Note

This routine is implemented as a macro.

### Example

```
#include "sled.h"

/* ... */
case GREYPLUS:
    sled_InsertRows(sed, sled_GetRow(sed), 1);
    sled_Repaint(sed);
    return(TRUE);
case GREYMINUS:
    sled_DeleteRows(sed, sled_GetRow(sed), 1);
    sled_Repaint(sed);
    return(TRUE);
}
```



### Synopsis

```
#include "sled.h"

void sled_SetColVar(sled, col, row, data);

    sed_type sled;      the sled
    int col;            the col
    int row;            the row
    VOID *data;         data for array entry
```

### Description

This routine copies the data pointed to by *data* into the sled's column array. The data is copied into column array *col* at row *row*.

This routine expects a VOID \* pointer; you must cast *data* to a VOID \*.

### Return Value

There is no return value.

### See Also

**sled\_GetColVar**

### Example

```
#include "sled.h"

int q = 7;
char *name = "Frog";

/* ... */

sled_SetColVar(sled, 0, 0, (VOID *) name);
sled_SetColVar(sled, 1, 0, (VOID *) &q);
```

## Synopsis

```
#include "slug.h"

int slug_Go(slug, start, row, col, data);

    sed_type slug;           the slug object
    int start;               starting field number
    int row;                 display row
    int col;                 display column
    VOID *data;              slug data pointer
```

## Description

This routine paints, activates, then pops a slug menuing system. *start* determines which menu choice is initially highlighted. *row* and *col* determine where the menu will be placed on the display. *data* is a data pointer that passes data to the user-supplied functions.

Consult the *C-scape Manual* for a detailed description of the slug menuing system.

## Return Value

Returns the value returned by the user function, the value specified in the definition structure, or 0 if **Esc** was pressed.

## See Also

**slug\_Open**, **sed\_Close**

## Example

```
#include "slug.h"

/* ... */
sed_type slug;

slug = slug_Open(menu_def, SLUG_VERTICAL, NULL);

/* ... */
slug_Go(slug, 0, 0, 0, (VOID *) my_data);
```

### Synopsis

```
#include "slug.h"

sed_type slug_Open(slug_def, dir_flag, border, bck, sel, bdr);

    struct slug_list slug_def[];  the slug definition structure
    int dir_flag;                 the direction flag
    bd_fptr border                the border function
    byte bck;                     the background color
    byte sel;                     the selected menu choice color
    byte bdr;                     the border color
```

### Description

This routine creates a slug menu object from a slug definition structure, *slug\_def*. *dir\_flag* determines the orientation of the slug menu. Use `SLUG_VERTICAL` for vertical menus and `SLUG_HORIZONTAL` for horizontal menus. *border* is the slug menu's border. *bck\_color*, *sel\_color*, and *bdr\_color* determine the colors of the slug menu.

Consult the *C-scape Manual* for a detailed description of the slug menuing system.

### Return Value

Returns a pointer to the new slug object. A NULL pointer values indicates insufficient memory or an error in the definition structure.

### Note

This routine is implemented as a macro.

### See Also

`slug_Go`, `sed_Close`

## Example

```
struct slug_list disk_menu[] = {

{ " Save ", "Save a file",  NULL, save, 0 },
{ " Load ", "Load a file",  NULL, load, 0 },
{  NULL,    "Disk menu",    NULL, NULL, 2 }
};

struct slug_list menu_def[] = {

{ " Disk ", "Disk commands", disk_menu, NULL, 0 },
{ " Quit  ", "Exit",          , NULL,      quit, 0 },
{  NULL,    "Main menu",      NULL,        NULL, 1 }
};

/* ... */

sed_type slug;

slug = slug_Open(menu_def, SLUG_VERTICAL, bd_cua, 0x07, 0x70,
0x07);

/* ... */

slug_Go(slug, 0, 0, 0, (VOID *) my_data);

/* ... */
```

### Synopsis

```
#include "slug.h"

void slug_Repaint(slug, row, col);

    sed_type slug;           the slug object
    int row;                 display row
    int col;                 display column
```

### Description

This routine paints a slug menu to the display. *row* and *col* determine where the menu will appear on the display.

When creating a 123-type menu, you should be call this before **slug\_Go**.

### Return Value

There is no return value.

### Example

```
/* ... */

sed_type slug;

slug = slug_Open(123_menu, SLUG_HORIZONTAL, NULL, 0x07, 0x70,
0x07);

slug_Repaint(slug, 0, 0);
slug_Go(slug, 0, 0, 0, (VOID *) my_data);

/* ... */
```

**Synopsis**

```
void spc_func(name);  
char *name;           special function to be prototyped
```

**Description**

This routine takes the name of a special function that you wish to prototype and performs that action.

**Return Value**

This routine has no return value.

**Example**

```
spc_func(spc_Zapata);  
  
main() {  
    /* ... */  
    sed_SetSpecial(sed, spc_Zapata);  
    /* ... */  
    sed_DoSpecial(sed, scancode);  
    /* ... */  
}
```

### Synopsis

```
#include "scancode.h"

boolean special_key(sed, scancode);

    sed_type sed;           the sed
    int scancode;           keystroke to process
```

### Description

This routine is used by the standard field function to handle function keys and other special keys in following manner:

FN1                Call **help\_Show**

MOU\_HERE        Return; another field was selected with the mouse.

MOU\_THERE    Call **sed\_ToggleExit**; another sed was selected with the mouse.

This routine can be altered to change the way the standard field functions handle function keys.

### Return Value

Returns TRUE if a keystroke was intercepted.

### See Also

**inter\_field**, **inter\_page**, **sed\_SetSpecial**

### Example

```
void string_fkey(sed)
    sed_type sed;
{
    int    scancode, key;

    scancode = kb_Read();

    if (special_key(sed, scancode))
        return;
    /* ... */
```

**Synopsis**

<code>char *strcenter(s, len);</code>	Center a string
<code>char *s;</code>	the string
<code>int len;</code>	length of centered string
<code>char *strclip(s)</code>	Remove trailing spaces from a string
<code>char *s;</code>	the string
<code>char *strfill(s, ch, count);</code>	Fill a string with characters
	the string
<code>char *s;</code>	character to fill with
<code>char ch;</code>	number of characters
<code>int count;</code>	
<code>char *strleft(s, len);</code>	Left justify a string
<code>char *s;</code>	the string
<code>int len;</code>	length of justified string.
<code>char *strpad(s, len);</code>	Pad a string with spaces
<code>char *s;</code>	the string
<code>int len;</code>	length of padded string.
<code>char *strpreclip(s);</code>	Remove leading spaces from a string
<code>char *s;</code>	the string
<code>char *strright(s, len);</code>	Right justify a string
<code>char *s;</code>	the string
<code>int len;</code>	length of justified string.



## Description

- strcenter** adjusts the string's trailing and leading spaces until it is *len* characters long with the text centered. The string must have space allocated for at least *len* + 1 characters.
- strclip** removes the trailing spaces from a string.
- strfill** fills string *s* with *count* characters *ch*. The string is '\0' terminated. The string must have enough space to contain at least *count* + 1 characters.
- strleft** removes the string's leading spaces and adjusts the trailing spaces until it is *len* characters long. The string must have space allocated for at least *len* + 1 characters.
- strpad** adjusts the string's trailing spaces until it is *len* characters long. The string must have space allocated for at least *len* + 1 characters.
- strright** removes the string's trailing spaces and adjusts the leading spaces until it is *len* characters long. The string must have space allocated for at least *len* + 1 characters.

## Return Value

All these routines return a pointer to the string passed to them.

## Examples

```
void main()                                STRCLIP OUTPUT:
{
    char s[11];
    strcpy(s, " abc      ");              [ abc      ]
    printf("[%s]\n", s);                   [ abc]
    strclip(s);
    printf("[%s]\n", s);
}
```

```
void main()
```

```
{
```

```
    char s[11];
```

```
    strcpy(s, "abc");
```

```
    printf("[%s]\n", s);
```

```
    strcenter(s, 10);
```

```
    printf("[%s]\n", s);
```

```
}
```

```
void main()
```

```
{
```

```
    char s[11];
```

```
    strfill(s, 'a', 8);
```

```
    printf("[%s]\n", s);
```

```
}
```

```
void main()
```

```
{
```

```
    char s[11];
```

```
    strcpy(s, "  abc  ");
```

```
    printf("[%s]\n", s);
```

```
    strleft(s, 10);
```

```
    printf("[%s]\n", s);
```

```
}
```

```
void main()
```

```
{
```

```
    char s[11];
```

```
    strcpy(s, "abc");
```

```
    printf("[%s]\n", s);
```

```
    strpad(s, 10);
```

```
    printf("[%s]\n", s);
```

```
}
```

STRCENTER OUTPUT:

```
[abc]
```

```
[  abc  ]
```

STRFILL OUTPUT:

```
[aaaaaaaa]
```

STRLEFT OUTPUT:

```
[  abc  ]
```

```
[abc     ]
```

STRPAD OUTPUT:

```
[abc]
```

```
[abc     ]
```

```
void main()
```

```
{
```

```
    char s[11];
```

```
    strcpy(s, "  abc  ");
```

```
    printf("[%s]\n", s);
```

```
    strpreclip(s, 10);
```

```
    printf("[%s]\n", s);
```

```
}
```

```
void main()
```

```
{
```

```
    char s[11];
```

```
    strcpy(s, "  abc  ");
```

```
    printf("[%s]\n", s);
```

```
    strright(s, 10);
```

```
    printf("[%s]\n", s);
```

```
}
```

STRPRECLIP OUTPUT:

```
[  abc  ]
```

```
[abc  ]
```

STRRIGHT OUTPUT:

```
[  abc  ]
```

```
[      abc]
```

**Synopsis**

```
char *strwrap(text, row, width);

char *text;           the string to word wrap
int *row;             number of rows in word-wrapped text
int width;           width of the text rows
```

**Description**

This routine allocates space for a string and copies the string *text* into it. If any of the lines of text in *text* are longer than *width*, **strwrap** splits them at the appropriate word break. *row* holds the number of rows in the new string. Rows are terminated by '\n's.

**Return Value**

Returns a pointer to the new word wrapped string. Returns NULL if unable to allocate space for the new string.

**Example**

```
void main()
{
    char *s;
    int r;

    s = strwrap("The quick red slug slimed over Josh.", &r, 15);
    printf("%s\n", s);

    printf("row = %d.\n", r);
}
```

**OUTPUT:**

```
The quick
red slug
slimed over the
frog.
```

```
row = 4.
```

### Synopsis

```
boolean ted_AddChar(sed, c);  
    sed_type sed;           the sed  
    char c;                 the character
```

### Description

This routine adds character *c* to the sed's text buffer at the current cursor location.

If the sed is in insert mode, the characters following the current cursor location are displaced to the right to accommodate the inserted character. If the sed is not in insert mode the character overwrites the character at the current cursor location.

If the sed is in refresh mode, the display is updated.

### Return Value

Returns TRUE if successful, FALSE if not.

### See Also

**ted\_AddRow**, **ted\_AddString**, **ted\_SetInsert**

### Example

```
switch (scancode) {  
  
    /* ... */  
  
    default:  
        key = ascii(scancode);  
  
        if ( isprint(key) ) {  
            ted_AddChar(sed, key);  
        }  
        break;  
}
```

### Synopsis

```
boolean ted_AddRow(sed);  
    sed_type sed;          the sed
```

### Description

This routine inserts a newline character in the sed at the current cursor location.

If the sed is in refresh mode, the display is updated.

### Return Value

Returns TRUE if successful, FALSE if not.

### See Also

**ted\_AddChar, ted\_DeleteChar**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
  
    /* ... */  
  
    case ENTER:  
        ted_AddRow(sed);  
        break;  
  
    /* ... */  
}
```

**Synopsis**

```
boolean ted_AddString(sed, string, len);  
  
    sed_type sed;           the sed  
    char *string;          the string  
    int len;               the length of the string
```

**Description**

This routine adds string *string* to the sed at the current cursor location. The cursor location is moved to the end of the string. *len* is the length of the string.

If the sed is in insert mode, the characters following the current cursor location are displaced to the right to accommodate the inserted characters. If the sed is not in insert mode the string overwrites the characters at the current cursor location.

If the sed is visible, the display is updated.

**Return Value**

Returns TRUE if successful, FALSE if not.

**See Also****ted\_AddChar****Example**

```
switch (scancode) {  
  
    /* ... */  
  
    case FN3: /* insert C keyword 'case:' */  
        ted_AddString(sed, "case:\t", 6);  
        break;  
  
    /* ... */  
}
```

**Synopsis**

```
boolean ted_BlockAttr(sed, attr);  
    sed_type sed;           the sed  
    byte attr;             the attribute
```

**Description**

This routine changes the display attribute of the current marked region in the sed to *attr*.

If there is no currently marked region the routine does nothing.

**Return Value**

Returns TRUE if successful, FALSE if not.

**See Also**

**ted\_SetMark**

**Note**

This routine is implemented as a macro.

**Example**

```
switch (scancode) {  
  
    /* ... */  
  
    case ALT_H:    /* highlight marked area */  
        ted_BlockAttr(sed, 0x70);  
        break;  
  
    /* ... */  
}
```



### Synopsis

```
boolean ted_BlockCopy(sed, buffer);  
    sed_type sed;           the sed  
    menu_type buffer;       the cut buffer
```

### Description

This routine copies the current marked region in the sed to *buffer*. The marking mode of the region is stored in the buffer. **ted\_BlockPaste** can later paste the buffer back into a sed.

If there is no currently marked region the routine does nothing.

This routine does not change the text buffer's marking mode.

### Return Value

Returns TRUE if successful, FALSE if not.

### See Also

**ted\_BlockCut**, **ted\_BlockPaste**, **ted\_SetMark**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
case GREYPLUS:  
    if ( ted_GetMark(sed) == TED_NOMARK) { /*copy current line */  
        ted_GoHome(sed);  
        ted_SetRefresh(sed, TED_NOREFRESH);  
        ted_SetMark(sed, TED_MARK);  
        ted_GoEnd(sed);  
    }  
    ted_SetRefresh(sed, TED_REFRESH);  
    ted_BlockCopy(sed, copy_menu);  
    ted_SetMark(sed, TED_NOMARK);  
    break;
```

### Synopsis

```
boolean ted_BlockCut(sed, buffer);  
    sed_type sed;           the sed  
    menu_type buffer;       the cut buffer
```

### Description

This routine deletes the current marked region in the sed and copies it to *buffer*; it also stores the marking mode of the region there. **ted\_BlockPaste** can later paste the buffer back into a sed.

If there is no currently marked region the routine does nothing.

If the sed is visible, the display is updated.

The text buffer's marking mode is reset to TED\_NOMARK.

### Return Value

Returns TRUE if successful, FALSE if not.

### See Also

**ted\_BlockCopy**, **ted\_BlockPaste**, **ted\_SetMark**

### Note

This routine is implemented as a macro.

### Example

```
case GREYMINUS:  
    if ( ted_GetMark(sed) == TED_NOMARK) { /*cut current line */  
        ted_GoHome(sed);  
        ted_SetRefresh(sed, TED_NOREFRESH);  
        ted_SetMark(sed, TED_MARK);  
        ted_GoEnd(sed);  
    }  
    ted_SetRefresh(sed, TED_REFRESH);  
    ted_BlockCut(sed, copy_menu);  
    ted_SetMark(sed, TED_NOMARK);  
    break;
```

### Synopsis

```
void ted_BlockDelete(sed);  
    sed_type sed;          the sed
```

### Description

This routine deletes the current marked region in the sed without copying it to a *buffer*. The contents of the marked region are lost and cannot be retrieved.

If there is no currently marked region the routine does nothing.

If the sed is in refresh mode, the display is updated.

The text buffer's marking mode is reset to TED\_NOMARK.

### Return Value

There is no return value.

### See Also

**ted\_BlockCopy**, **ted\_BlockPaste**, **ted\_SetMark**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
case ALT_D:  
    if ( ted_GetMark(sed) == TED_NOMARK ) {  
        sed_BorderPrompt(sed, "No current marked region.");  
    }  
    else {  
        sed_BorderPrompt(sed, "Okay to delete marked region?");  
        if ( kb_Read() == 'Y' ) {  
            ted_BlockDelete(sed);  
        }  
    }  
    break;
```

### Synopsis

```
boolean ted_BlockPaste(sed, buffer);  
    sed_type sed;           the sed  
    menu_type buffer;       the paste buffer
```

### Description

This routine pastes the paste *buffer* into the sed at the current cursor location according to the marking mode of the block. If the block is in column mode, **ted\_BlockPaste** pastes it column-wise; otherwise **ted\_BlockPaste** pastes it directly.

The paste buffer's contents are usually set with **ted\_BlockCopy** or **ted\_BlockCut**. Any menu, however, can be used as a paste buffer.

If the sed is in insert mode, **ted\_BlockPaste** inserts the block; otherwise it writes over existing text.

If the sed is visible, the display is updated.

### Return Value

Returns TRUE if successful, FALSE if not.

### See Also

**ted\_BlockCopy**, **ted\_BlockCut**, **ted\_SetMark**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
case INS:  
    ted_BlockPaste(sed, copy_menu);  
    break;  
  
/* ... */  
}
```

### Synopsis

```
char ted_DeleteChar(sed);  
    sed_type sed;           the sed
```

### Description

This routine deletes the character at the sed's current cursor location. The characters following the current cursor location move to the left to fill in the gap left by the deleted character.

If the sed is visible, the display is updated.

### Return Value

Returns the deleted character.

### See Also

**ted\_BlockDelete, ted\_AddChar**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
  
    /* ... */  
  
    case BACKSPACE:  
        if ( ted_LeftChar(sed, TED_TABJUMP) ) {  
            ted_DeleteChar(sed);  
        }  
        break;  
  
    /* ... */  
}
```

### Synopsis

```
int ted_DownChar(sed);  
    sed_type sed;          the sed
```

### Description

This routine moves the cursor down to the next row in the sed.

For example, **ted\_DownChar** would have the following effect:

*Before:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

*After:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

### Return Value

This routine returns TRUE if it succeeded in moving the cursor, otherwise it returns FALSE.

### See Also

**ted\_SetMoveMethod**, **ted\_LeftChar**, **ted\_RightChar**, **ted\_UpChar**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
    case DOWN:  
        ted_DownChar(sed);  
        break;  
  
    /* ... */  
}
```

### Synopsis

```
long ted_GetCursor(sed);  
    sed_type sed;          the sed
```

### Description

This routine returns the offset of the character that the cursor is currently nearest to. For example, in box 1 **ted\_GetCursor** would return 5, in box 2 **ted\_GetCursor** would return 30:

1:

I am the atomic powered robot.  
Please give my regards to  
everybody.

2:

I am the atomic powered robot.  
Please give my regards to  
everybody.

### Return Value

Returns a long value corresponding to the offset of the cursor.

### See Also

**ted\_GotoCursor**, **ted\_LeftChar**, **ted\_RightChar**

### Example

```
long cursor;  
char msg[200];  
  
/* ... */  
switch (scancode) {  
case ALT_W: /* display the length of the next word */  
    ted_RightWord(sed);  
    cursor = ted_GetCursor(sed);  
    ted_RightWord(sed);  
    cursor = ted_GetCursor(sed) - cursor;  
    sprintf(msg, "Word is %ld characters long.", cursor);  
    sed_BorderPrompt(sed, "msg");  
    break;
```

**Synopsis**

```
int ted_GetInsert(sed);  
    sed_type sed;          the sed
```

**Description**

This routine gets the insert mode of the sed.

The insert mode can have the following values:

**TED\_INSERT**            Insert mode. Characters are slid to the left when new characters are written to the sed.

**TED\_OVERWRITE**        Overwrite mode. New characters are written on top of the old characters in the sed.

**Return Value**

Returns **TED\_INSERT** if the sed is in insert mode and **TED\_OVERWRITE** if the sed is in overwrite mode.

**See Also**

**ted\_SetInsert**

**Note**

This routine is implemented as a macro.

**Example**

```
switch (scancode) {  
case INS:  
    if ( ted_GetInsert(sed) == TED_OVERWRITE ) {  
        ted_SetInsert(sed, TED_INSERT);  
    }  
    else {  
        ted_SetInsert(sed, TED_OVERWRITE);  
    }  
    break;
```



**Synopsis**

```
int ted_GetLineLen(sed);  
    sed_type sed;          the sed
```

**Description**

This routine returns the length in characters of the line on which the cursor is currently placed. The length includes the terminating newline character ('`\n`') if there is one. The length is not necessarily the displayed length, as tabs ('`\t`') count as only one character.

**Return Value**

Returns the current line length.

**See Also**

**ted\_GetString**

**Note**

This routine is implemented as a macro.

**Example**

```
char msg[81];  
  
/* ... */  
  
switch (scancode) {  
  
/* ... */  
  
case ALT_L:  
    sprintf(msg, "Line length = %d characters.",  
            ted_GetLineLen(sed));  
    sed_BorderPrompt(sed, msg);  
    break;  
/* ... */  
}
```

**Synopsis**

```
int ted_GetMark(sed);  
    sed_type sed;           the sed
```

**Description**

This routine returns the current marking status of the sed. The various block operations, such as **ted\_BlockCopy**, use marked regions.

**Return Value**

Returns the current marking status of the sed. The marking status is one the following values:

TED_NOMARK	Marking mode turned off.
TED_MARK	Normal marking mode.
TED_COLMARK	Column-wise marking mode.
TED_FIXMARK	Marked region is fixed.

**See Also****ted\_SetMark****Example**

```
switch (scancode) {  
case GREYPLUS:  
    if ( ted_GetMark(sed) == TED_NOMARK) { /*copy current line */  
        ted_GoHome(sed);  
        if ( ted_GetRefresh(sed) == TED_REFRESH ) {  
            ted_SetRefresh(sed, TED_NOREFRESH); /* don't show mark */  
        }  
        ted_SetMark(sed, TED_MARK);  
        ted_GoEnd(sed);  
    }  
}
```

## ted\_GetMaxSize

Get the text buffer's maximum size

---

### Synopsis

```
long ted_GetMaxSize(sed);  
    sed_type sed;           the sed
```

### Description

This routine gets the sed's maximum text buffer size.

The maximum size is used for limiting the size of a text buffer. This is useful when editing a note pad of a specific size.

### Return Value

Returns the maximum size of the sed. This value is 0L if the text buffer has no maximum size.

### See Also

**ted\_SetMaxSize**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
  
    /* ... */  
  
    case ALT_I:    /* increase maxsize */  
        ted_SetMaxSize(sed, ted_GetMaxSize(sed) + 100L);  
        break;  
  
    /* ... */  
}
```

### Synopsis

```
char ted_GetNewLineChar(sed);  
    sed_type sed;          the sed
```

### Description

This routine gets the character currently used to display newlines in the sed. The default newline character is a space.

### Return Value

Returns the character used to display newlines in the sed

### See Also

**ted\_SetNewLineChar**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
  
    /* ... */  
  
    case CTRL_N:  
        if ( ted_GetNewLineChar(sed) == ' ' ) {  
            ted_SetNewLineChar(sed, '<');  
        }  
        else {  
            ted_SetNewLineChar(sed, ' ');  
        }  
        sed_Update(sed);  
        break;  
  
    /* ... */  
}
```

### Synopsis

```
void ted_GetPosition(sed, row, col);

    sed_type sed;           the sed
    int *row;               the position row
    int *col;               the position col
```

### Description

This routine ascertains the current text cursor position of the sed and stores it in the locations to which *row* and *col* point.

The cursor position can later be used by **ted\_GotoPosition** to restore the cursor to a previous location.

### Return Value

There is no return value.

### See Also

#### ted\_GotoPosition

### Example

```
int mem_row = 0, mem_col = 0;

/* ... */

switch (scancode) {
case CTRL_M:      /* remember current position */
    ted_GetPosition(sed, &mem_row, &mem_col);
    break;

case CTRL_R:      /* return to remembered position */
    ted_GotoPosition(sed, mem_row, mem_col);
    break;

/* ... */
}
```

### Synopsis

```
int ted_GetRefresh(sed);  
    sed_type sed;           the sed
```

### Description

Gets the refresh mode of the sed. The refresh mode determines whether the sed will be repainted after a **ted\_** operation or not.

The refresh mode can have one of the following values:

<b>TED_REFRESH</b>	Update the sed's displayed image after each <b>ted_</b> operation.
<b>TED_NOREFRESH</b>	Do not update the sed's displayed image after each <b>ted_</b> operation.

Refreshing is usually disabled for operations such as global search and replace where it is not desirable to show all the changes taking place.

### Return Value

Returns the current refresh mode of the sed, either **TED\_REFRESH** or **TED\_NOREFRESH**.

### See Also

**ted\_SetRefresh**

### Note

This routine is implemented as a macro.

## Example

```
switch (scancode) {

/* ... */

case GREYPLUS:
    if ( ted_GetMark(sed) == TED_NOMARK) { /* copy current line */
        ted_GoHome(sed);

        if ( ted_GetRefresh(sed) == TED_REFRESH ) {
            ted_SetRefresh(sed, TED_NOREFRESH); /* don't show mark */
        }
        ted_SetMark(sed, TED_MARK);
        ted_GoEnd(sed);
    }

    ted_SetRefresh(sed, TED_REFRESH);
    ted_BlockCopy(sed, copy_menu);
    ted_SetMark(sed, TED_NOMARK);
    break;

/* ... */
}
```

### Synopsis

```
long ted_GetSize(sed);  
    sed_type sed;          the sed
```

### Description

This routine returns the current count of characters in the text buffer.

### Return Value

Returns the number of characters in the text buffer.

### See Also

**ted\_GetMaxSize, ted\_SetMaxSize**

### Note

This routine is implemented as a macro.

### Example

```
char msg[81];  
  
/* ... */  
  
switch (scancode) {  
  
case ALT_S:  
    sprintf(msg, "Text size = %ld characters.", ted_GetSize(sed));  
    sed_BorderPrompt(sed, msg);  
    break;  
  
/* ... */  
}
```



**Synopsis**

```
int ted_GetString(sed, string, len);

    sed_type sed;           the sed
    char *string;           buffer to copy the string into
    int len;                number of characters to copy
```

**Description**

This routine copies up to *len* characters starting at the current cursor location into string. **string** should be at least one character longer than **len**.

If there are less than *len* characters following the current cursor location, **ted\_GetString** copies the remaining characters and returns the number actually copied.

**Return Value**

Returns the number of characters actually copied.

**See Also****ted\_AddString****Example**

```
char buf[200];
int len;

/* ... */
case ALT_K:                /* kill the current line */
    len = ted_GetLineLen(sed);
    ted_GoHome(sed);
    ted_GetString(sed, buf, len); /* save it for later */
    ted_SetRefresh(sed, TED_NOREFRESH);
    ted_Mark(sed, TED_MARK);
    ted_GoEnd(sed);
    ted_SetRefresh(sed, TED_REFRESH);
    ted_BlockDelete(sed);
    break;
```

**Synopsis**

```
int ted_GetTabChar(sed);  
    sed_type sed;          the sed
```

**Description**

This routine gets the character currently used to display tabs in the sed. The default tab character is a space.

**Return Value**

Returns the character currently used to display tabs in the sed.

**See Also**

**ted\_SetTabChar**

**Note**

This routine is implemented as a macro.

**Example**

```
switch (scancode) {  
  
    /* ... */  
  
    case CTRL_T:  
        if ( ted_GetTabChar(sed) == ' ' ) {  
            ted_SetTabChar(sed, '>');  
        }  
        else {  
            ted_SetTabChar(sed, ' ');  
        }  
        sed_Update(sed);  
        break;  
  
    /* ... */  
}
```

### Synopsis

```
int ted_GetTabSize(sed);  
    sed_type sed;          the sed
```

### Description

This routine gets the sed's tab size.

The tab size determines the display width of tabs.

### Return Value

Returns the tab size of the sed.

### See Also

**ted\_SetTabSize**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
  
    /* ... */  
  
    case ALT_T:  
        if ( ted_GetTabSize(sed) == 8 ) {  
            ted_SetTabSize(sed, 4);          /* small tabs */  
        }  
        else {  
            ted_SetTabSize(sed, 8);          /* big tabs */  
        }  
        sed_Update(sed);  
        break;  
  
    /* ... */  
}
```

### Synopsis

```
int ted_GetWrapWidth(sed);  
    sed_type sed;          the sed
```

### Description

This routine gets the sed's wrap width.

The wrap width is the maximum width of line in the text buffer. If a line is longer than the wrap width it is word wrapped at the first space character before the end of the line.

The default wrap width is 32000.

### Return Value

Returns the wrap width of the sed.

### See Also

**ted\_SetWrapWidth**

### Note

This routine is implemented as a macro.

### Example

```
int wrap_width;  
  
/* ... */  
  
switch (scancode) {  
case ALT_W:  
    wrap_width = ted_GetWrapWidth(sed);  
  
    if ( get_int_sed("Wrap Width: ", &wrap_width) ) {  
        ted_SetWrapWidth(sed, wrap_width);  
        sed_Update(sed);  
    }  
    break;
```

## ted\_GoBottom

Move to the bottom of the sed text

---

### Synopsis

```
boolean ted_GoBottom(sed);  
    sed_type sed;          the sed
```

### Description

This routine moves the text cursor to the last character in the sed.

### Return Value

This routine returns TRUE if it succeeds in moving the cursor; otherwise, it returns FALSE.

### See Also

**ted\_GoTop**

### Example

```
switch (scancode) {  
  
    /* ... */  
  
    case CTRL_PGDN:  
        ted_GoBottom(sed);  
        break;  
  
    /* ... */  
}
```

**Synopsis**

```
boolean ted_GoEnd(sed);  
    sed_type sed;          the sed
```

**Description**

This routine moves the text cursor to the end of the current row.

For example, **ted\_GoEnd** would have the following effect:

*Before:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

*After:*

I am the atomic powered robot.  
Please give my regards to\_  
everybody.

**Return Value**

This routine returns TRUE if it succeeded in moving the cursor, otherwise it returns FALSE.

**See Also**

**ted\_GoHome**

**Example**

```
switch (scancode) {  
  
    /* ... */  
  
    case END:  
        ted_GoEnd(sed);  
        break;  
  
    /* ... */  
}
```

## ted\_GoHome

Move to the start of the row

---

### Synopsis

```
boolean ted_GoHome(sed);  
    sed_type sed;          the sed
```

### Description

This routine moves the text cursor to the start of the current row.

For example, **ted\_GoHome** would have the following effect:

*Before:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

*After:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

### Return Value

This routine returns TRUE if it succeeded in moving the cursor, otherwise it returns FALSE.

### See Also

**ted\_GoEnd**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
case HOME:  
    ted_GoHome(sed);  
    break;  
  
/* ... */  
}
```

## Synopsis

```
boolean ted_GotoCursor(sed, offset);  
  
    sed_type sed;           the sed  
    long offset;           offset to place cursor at
```

## Description

This routine places the cursor at the *offset*th character in *sed*'s text buffer. The cursor will always end up in a location containing text and will not be moved past the end of the text.

For example,

```
ted_GotoCursor(sed, 7);
```

would have the following effect:

*Before:*

I I am the atomic powered robot.  
Please give my regards to  
everybody.

*After:*

I am the the atomic powered robot.  
Please give my regards to  
everybody.

## Return Value

Returns TRUE if successful, FALSE if not.

## See Also

**ted\_LeftChar, ted\_RightChar, ted\_GetCursor**

## Example

```
case ALT_R:  
    while ( ted_Search(sed, search_str, TED_FORWARD) ) {  
        sed_BorderPrompt(sed, "Replace? Y, N ");  
        if ( kb_Read() == 'N' ) {  
            /* move past the current occurrence of search */  
            ted_GotoCursor(sed, ted_GetCursor(sed) + 1L);  
        }  
    }
```



### Synopsis

```
boolean ted_GoTop(sed);  
    sed_type sed;          the sed
```

### Description

This routine moves the text cursor to the first character in the sed.

### Return Value

This routine returns TRUE if it succeeded in moving the cursor, otherwise it returns FALSE.

### See Also

**ted\_GoBottom**

### Note

This routine is implemented as a macro.

### Example

```
switch (scancode) {  
  
    /* ... */  
  
    case CTRL_PGUP:  
        ted_GoTop(sed);  
        break;  
  
    /* ... */  
}
```

### Synopsis

```
boolean ted_GotoPosition(sed, row, col);  
  
    sed_type sed;           the sed  
    int row;                the position row  
    int col;                the position col
```

### Description

This routine moves the text cursor to the position specified by *row* and *col*.

This routine is useful for restoring the cursor to a position previously obtained with **ted\_GetPosition**.

### Return Value

This routine returns TRUE if it succeeded in moving the cursor, otherwise it returns FALSE.

### See Also

**ted\_GetPosition**

### Note

This routine is implemented as a macro.

### Example

```
int mem_row = 0, mem_col = 0;  
  
/* ... */  
switch (scancode) {  
case CTRL_M:      /* remember current position */  
    ted_GetPosition(sed, &mem_row, &mem_col);  
    break;  
  
case CTRL_R:      /* return to remembered position */  
    ted_GotoPosition(sed, mem_row, mem_col);  
    break;
```

## ted\_LeftChar

Move left one character

### Synopsis

```
boolean ted_LeftChar(sed, mode);  
    sed_type sed;           the sed  
    int mode;               the movement mode
```

### Description

This routine moves the cursor one character to the left. If the cursor is at the start of a line, it moves to the end of the previous row.

The movement mode determines how **ted\_LeftChar** treats tabs and can be one of the following values:

**TED\_TABJUMP**            The cursor skips over tabs.

**TED\_NOTABJUMP**        The cursor advances within tabs.

For example, **ted\_LeftChar** would have the following effect:

*Before:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

*After:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

### Return Value

This routine returns TRUE if it succeeded in moving the cursor, otherwise it returns FALSE.

### See Also

**ted\_SetMoveMethod**, **ted\_DownChar**, **ted\_RightChar**, **ted\_UpChar**

### Note

This routine is implemented as a macro.

## Example

```
int mode = TED_TABJUMP;

/* ... */

switch (scancode) {

/* ... */

case CTRL_M:
    if ( mode == TED_TABJUMP ) {
        mode = TED_NOTABJUMP; /* editor cursor movement */
    }
    else {
        mode = TED_TABJUMP; /* word processor cursor movement */
    }
    break;

case LEFT:
    ted_LeftChar(sed, mode);
    break;

case BACKSPACE:
    if ( ted_LeftChar(sed, TED_TABJUMP) ) {
        ted_DeleteChar(sed);
    }
    break;

/* ... */
}
```

## ted\_LeftWord

Move left one word

---

### Synopsis

```
boolean ted_LeftWord(sed);  
    sed_type sed;          the sed
```

### Description

This routine moves the cursor the start of the next word to the left.

The start of the next word is defined to be the first non whitespace (*tab*, *space*, or *newline*) character following the next whitespace character(s).

For example, **ted\_LeftWord** would have the following effect:

*Before:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

*After:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

### Return Value

This routine returns TRUE if it succeeded in moving the cursor, otherwise it returns FALSE.

### Note

This routine is implemented as a macro.

### See Also

**ted\_RightWord**

## Example

```
switch (scancode) {  
  
    /* ... */  
  
    case ALT_P:      /* go to previous word */  
        ted_LeftWord(sed);  
        break;  
  
    /* ... */  
}
```

### Synopsis

```
void ted_PageDown(sed);  
    sed_type sed;          the sed
```

### Description

This routine moves the text down one page leaving the cursor in the same position on the display. Some cursor move methods (**ted\_Follow**) may not allow this if you place the cursor past the end of the text.

### Return Value

There is no return value.

### See Also

**ted\_PageUp**

### Example

```
switch (scancode) {  
  
    /* ... */  
  
    case PGDN:  
        ted_PageDown(sed);  
        break;  
  
    /* ... */  
}
```

### Synopsis

```
void ted_PageUp(sed);  
    sed_type sed;          the sed
```

### Description

This routine moves the text up one or less pages leaving the cursor in the same position on the display. If the text is not scrolled down then the cursor is moved to the top of the text.

### Return Value

There is no return value.

### See Also

### ted\_PageDown

### Example

```
switch (scancode) {  
  
    /* ... */  
  
    case PGUP:  
        ted_PageUp(sed);  
        break;  
  
    /* ... */  
}
```



## ted\_ReadFile

Fill a text buffer from a file

---

### Synopsis

```
long ted_ReadFile(sed, fp);  
    sed_type sed;           the sed  
    FILE *fp;               the file stream to read from
```

### Description

This routine reads the contents of file *fp* into the text buffer. It returns the number of characters read from the file.

**ted\_ReadFile** clears the current contents of the text buffer before it reads the file.

You must open the file before calling **ted\_ReadFile**. Text files should be opened in text mode; binary files should be opened in binary mode.

### Return Value

If it is successful this routine returns the number of characters read, otherwise it returns -1L.

### See Also

**ted\_WriteFile**

## Example

```
char file_name[200];
FILE *fp;

/* ... */
switch (scancode) {
case ALT_F:
    if ( get_string_sed("File to Read: ", file_name) ) {

        fp = fopen(file_name, "r");
        ted_ReadFile(sed, fp);
        sed_Update(sed);
        /* put it on the display for viewing */

        fclose(fp);
    }
    break;
/* ... */
```

## ted\_RightChar

Move right one character

---

### Synopsis

```
boolean ted_RightChar(sed, mode);  
    sed_type sed;           the sed  
    int mode;               the movement mode
```

### Description

This routine moves the cursor one character to the right. If the cursor is at the end of a line it is moved to the beginning end of the next row.

The movement mode determines how **ted\_RightChar** treats tabs and can be one of the following values:

TED_TABJUMP	The cursor skips over tabs.
TED_NOTABJUMP	The cursor advances within tabs.

For example, **ted\_RightChar** would have the following effect:

*Before:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

*After:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

### Return Value

This routine returns TRUE if it succeeded in moving the cursor, otherwise it returns FALSE.

### See Also

**ted\_SetMoveMethod**, **ted\_DownChar**, **ted\_LeftChar**, **ted\_UpChar**

### Note

This routine is implemented as a macro.

## Example

```
int mode = TED_TABJUMP;

/* ... */

switch (scancode) {

/* ... */

case CTRL_M:
    if ( mode == TED_TABJUMP) {
        mode = TED_NOTABJUMP; /* editor cursor movement */
    }
    else {
        mode = TED_TABJUMP; /* word processor cursor movement */
    }
    break;

case RIGHT:
    ted_RightChar(sed, mode);
    break;

/* ... */
}
```

## ted\_RightWord

Move right one word

---

### Synopsis

```
boolean ted_RightWord(sed);  
    sed_type sed;           the sed
```

### Description

This routine moves the cursor the start of the next word to the right.

The start of the next word is defined to be the first non whitespace (*tab, space, or newline*) character following the closest whitespace character(s) to the right.

For example, **ted\_RightWord** would have the following effect:

*Before:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

*After:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

### Return Value

This routine returns TRUE if it succeeded in moving the cursor, otherwise it returns FALSE.

### See Also

**ted\_LeftWord**

### Note

This routine is implemented as a macro.

### Example

```
case ALT_N:           /* go to next word */  
    ted_RightWord(sed);  
    break;
```

**Synopsis**

```
boolean ted_Search(sed, pattern, mode);  
    sed_type sed;           the sed  
    char *pattern;         the search pattern  
    unsigned mode;         the search mode
```

**Description**

This routine searches the sed text for *pattern*. It returns TRUE if it finds the pattern and then moves the cursor according to the search mode.

The search pattern is simply a character string containing the pattern for the routine searches.

The search mode determines the following actions, the search direction, the cursor position after the search, the search case sensitivity, and whether or not to delete the pattern when it is found.

The search direction is either forward or backward. The direction is specified by the using the TED\_FORWARD or TED\_BACKWARD mode.

The cursor normally moves to the start of the found pattern but it can be specified to move to the position one character past or before the pattern using the TED\_AFTER and TED\_BEFORE modes. This is useful for repeated searches.

By default, the search will be case sensitive. To search in a case insensitive manor, specify the TED\_INSENSITIVE mode.

You can specify the search mode combining the following values together with the | (OR) operator.

TED_FORWARD	Search forward from the current cursor location.
TED_BACKWARD	Search backward from the current cursor location.
TED_AFTER	Move the cursor one location past the found pattern.
TED_BEFORE	Move the cursor one location before the pattern.
TED_INSENSITIVE	Search without case sensitivity.

For example, the following code specifies a forward search for the word “avoidupois” in which the cursor is to be moved past the end of the pattern:

```
ted_Search(sed, "avoidupois", TED_FORWARD | TED_AFTER);
```

This search would have the following effect:

*Before:*

Is that measured in avoidupois or  
troy ounces?

*After:*

Is that measured in avoidupois\_or  
troy ounces?

## Return Value

Returns TRUE if the search was successful, FALSE otherwise.

## See Also

## Example

```
char search[200];

/* ... */

switch (scancode) {

/* ... */

case ALT_S:
    if ( get_string_sed("Search Pattern: ", search) ) {

        if ( !ted_Search(sed, search, TED_FORWARD | TED_AFTER ) ) {

            sed_BorderPrompt(sed, "Pattern not found");
        }
    }
    break;

/* ... */
}
```

## Synopsis

```
void ted_SetInsert(sed, mode);  
    sed_type sed;           the sed  
    boolean mode;          the insert mode
```

## Description

This routine sets the insert mode of the sed to *mode*. The insert mode determines how text is added to the sed.

The insert mode can have the following values:

TED_INSERT	Insert mode. Characters are slid to the right when new characters are written to the sed.
TED_OVERWRITE	Overwrite mode. New characters are written on top of the old characters in the sed.

The default insert mode is TED\_OVERWRITE.

The insert mode affects the operation of **menu\_Printf**. For normal operation, the insert mode should be set to TED\_OVERWRITE when using **menu\_Printf** to add text to a text buffer.

## Return Value

There is no return value.

## See Also

**ted\_GetInsert**

## Note

This routine is implemented as a macro.

## Example

```
if ( ted_GetInsert(sed) == TED_OVERWRITE ) {  
    ted_SetInsert(sed, TED_INSERT);  
}
```



### Synopsis

```
void ted_SetMark(sed, mode);  
  
    sed_type sed;           the sed  
    int mode;               the marking mode
```

### Description

This routine sets the marking status of the sed. The status is specified by *mode* and can have one of the following values:

- |             |   |
|-------------|---|
| TED_NOMARK  | Turn off marking.                                   |
| TED_MARK    | Turn on marking.                                    |
| TED_COLMARK | Turn on column-wise marking.                        |
| TED_FIXMARK | Fix the marked region as the current marked region. |

Block operations, such as **ted\_BlockCopy**, use the marked region.

The marked region consists of the text between the current cursor location and the cursor location when marking was turned on (the anchor). Turning on marking when the sed is already marked turns off the previous marking.

If the marking mode is set to TED\_FIXMARK the marked region is “fixed” between the current cursor location (the cleat) and the anchor.

If the sed is in refresh mode, the marked region appears in the sed’s highlight color.

### Return Value

There is no return value.

### See Also

**sed\_SetColors**, **ted\_BlockCopy**, **ted\_BlockCut**, **ted\_GetMark**

## Example

```
switch (scancode) {

/* ... */

case ALT_M:
    if ( ted_GetMark(sed) != TED_NOMARK ) {
        ted_SetMark(sed, TED_NOMARK);
    }
    else {
        ted_SetMark(sed, TED_MARK);
    }
    break;

case ALT_C:
    if ( ted_GetMark(sed) != TED_NOMARK ) {
        ted_SetMark(sed, TED_NOMARK);
    }
    else {
        ted_SetMark(sed, TED_COLMARK);
    }
    break;

case ALT_F:
    if ( ted_GetMark(sed) != TED_NOMARK ) {
        ted_SetMark(sed, TED_FIXMARK);
    }
    break;

/* ... */
}
```

**Synopsis**

```
void ted_SetMaxSize(sed, maxsize);  
    sed_type sed;           the sed  
    long maxsize;          the maximum size
```

**Description**

This routine sets the sed's maximum text buffer size to *maxsize*.

The maximum size limits the size of a text buffer. You cannot add more characters to the text buffer once it has reached its maximum size. This is useful when you want to edit a note pad of a specific size.

By default the text buffer is of unlimited size. To reset the text buffer to unlimited size, set *maxsize* equal to 0L.

You must set the maximum size of the text buffer when you use **ted\_funcs**.

**Return Value**

There is no return value.

**See Also**

**ted\_GetMaxSize**

**Note**

This routine is implemented as a macro.

**Example**

```
switch (scancode) {  
case ALT_I:    /* increase maxsize */  
    ted_SetMaxSize(sed, ted_GetMaxSize(sed) + 100L);  
    break;  
  
/* ... */
```

### Synopsis

```
void ted_SetMoveMethod(sed, move);  
    sed_type sed;           the sed  
    move_method move;      the cursor movement method
```

### Description

This routine sets the method by which the text cursor moves. The movement method is a pointer to a cursor movement function. The various **ted\_** cursor movement routines use the cursor movement function to determine how to move the cursor. The available cursor movement methods are:

**ted\_Past**      Allows the cursor to move past the end of lines. Moves up and down in a straight line.

**ted\_Follow**   Does not allow cursor motion past the end of lines. Up and down motion follows the ends of the lines. Motion is not allowed past the end of text.

The default movement method is **ted\_Follow**.

### Return Value

There is no return value.

### See Also

**ted\_DownChar**, **ted\_LeftChar**, **ted\_RightChar**, **ted\_UpChar**

## Example

```
int mode = TED_TABJUMP;

/* ... */

switch (scancode) {

/* ... */

case CTRL_M:
    if ( mode == TED_TABJUMP ) {

        mode = TED_NOTABJUMP; /* editor cursor movement */
        ted_SetMoveMethod(sed, ted_Past);
    }
    else {
        mode = TED_TABJUMP; /* word processor cursor movement */
        ted_SetMoveMethod(sed, ted_Follow);
    }
    break;

/* ... */
}
```

### Synopsis

```
void ted_SetNewLineChar(sed, nl);  
    sed_type sed;           the sed  
    int nl;                 the displayed newline character
```

### Description

This routine sets the character used to display newlines in the sed. The default newline character is a space. This routine does *not* update the display.

For example,

```
ted_SetNewlineChar(sed, '<');  
sed_Repaint(sed);
```

would have the following effect:

*Before:*

```
Hokkaido  
Honshu  
Iwo Jima  
Kyushu  
Okinawa  
Shikoku
```

*After:*

```
Hokkaido<  
Honshu<  
Iwo Jima<  
Kyushu<  
Okinawa<  
Shikoku<
```

### Return Value

There is no return value.

### See Also

**ted\_GetNewLineChar**

### Note

This routine is implemented as a macro.

## Example

```
switch (scancode) {  
  
    /* ... */  
  
    case CTRL_N:  
        if ( ted_GetNewLineChar(sed) == ' ' ) {  
            ted_SetNewLineChar(sed, '<');  
        }  
        else {  
            ted_SetNewLineChar(sed, ' ');  
        }  
        sed_Update(sed);  
        break;  
  
    /* ... */  
}
```

### Synopsis

```
void ted_SetRefresh(sed, mode);  
    sed_type sed;           the sed  
    int mode;               the refresh mode
```

### Description

Sets the refresh mode of the sed. The refresh mode determines whether the sed should be repainted after a **ted\_** operation or not.

The refresh mode can have one of the following values:

<b>TED_REFRESH</b>	Update the sed's displayed image after each <b>ted_</b> operation.
<b>TED_NOREFRESH</b>	Do not update the sed's displayed image after each <b>ted_</b> operation.

Refreshing is usually turned off for operations such as global search and replace where it is not desirable to show all the changes taking place.

### Return Value

There is no return value.

### See Also

**ted\_GetRefresh**

### Note

This routine is implemented as a macro.



## Example

```
char buf[200];
int len;

/* ... */

switch (scancode) {

/* ... */

case ALT_K:                /* kill the current line */

    len = ted_GetLineLen(sed);
    ted_GoHome(sed);
    ted_GetString(sed, buf, len);    /* save it for later */

    ted_SetRefresh(sed, TED_NOREFRESH);    /* don't show the mark */

    ted_Mark(sed, TED_MARK);
    ted_GoEnd(sed);

    /* show the delete and continue */
    ted_SetRefresh(sed, TED_REFRESH);

    ted_BlockDelete(sed);
    break;

/* ... */
}
```

### Synopsis

```
void ted_SetTabChar(sed, tab);  
  
    sed_type sed;           the sed  
    int tab;                the displayed tab character
```

### Description

This routine sets the character used to display tabs in the sed. The default tab character is a space. This routine does *not* update the display.

For example,

```
ted_SetTabChar(sed, '>');  
sed_Repaint(sed);
```

would have the following effect:

*Before:*

```
for (i = 0; i < 20; i++) {  
    if (flag) {  
        break;  
    }  
}
```

*After:*

```
for (i = 0; i < 20; i++) {  
> if (flag) {  
> > break;  
> }  
}
```

### Return Value

There is no return value.

### See Also

**ted\_GetTabChar**

### Note

This routine is implemented as a macro.

### Example

```
ted_SetTabChar(sed, '>');  
sed_Update(sed);
```

### Synopsis

```
void ted_SetTabSize(sed, tsize);  
    sed_type sed;           the sed  
    int  tsize;             the new tab size
```

### Description

This routine sets the sed's tab size.

The tab size determines the display width of tabs.

This routine refreshes the display unless the sed is not in refresh mode.

### Return Value

There is no return value.

### See Also

**ted\_SetRefresh, ted\_SetTabSize**

### Note

This routine is implemented as a macro.

### Example

```
int tab_size;  
  
/* ... */  
  
switch (scancode) {  
case ALT_T:  
    tab_size = ted_GetTabSize(sed);  
  
    if ( get_int_sed("Tab Size: ", &tab_size) ) {  
        ted_SetTabSize(sed, tab_size);  
        sed_Update(sed);  
    }  
    break;
```

### Synopsis

```
void ted_SetWrapWidth(sed, wwidth);

    sed_type sed;          the sed
    int wwidth;            the wrap width
```

### Description

This routine sets the sed's wrap width to **wwidth**.

The wrap width is the maximum width of line in the text buffer. If a line is longer than the wrap width it is word wrapped at the first space character before the end of the line.

The default wrap width is 32000.

### Return Value

There is no return value.

### See Also

#### ted\_GetWrapWidth

### Note

This routine is implemented as a macro.

### Example

```
int wrap_width;

/* ... */
switch (scancode) {
case ALT_W:
    wrap_width = ted_GetWrapWidth(sed);

    if ( get_int_sed("Wrap Width: ", &wrap_width) ) {
        ted_SetWrapWidth(sed, wrap_width);
        sed_Update(sed);
    }
    break;
```

### Synopsis

```
void ted_StartWorking(sed);  
    sed_type sed;           the sed
```

### Description

This routine places a sed into text editing mode. You must call **ted\_StartWorking** before you can use a sed as a text editor. **ted\_funcs** calls this routine in its **fenter** function.

### Return Value

There is no return value.

### Example

```
void ted_fenter(sed)  
    sed_type sed;  
{  
    std_fenter(sed);  
    ted_StartWorking(sed);           /* set up the cursor */  
}
```

## Synopsis

```
boolean ted_UpChar(sed);  
    sed_type sed;          the sed
```

## Description

This routine moves the cursor up the previous row in the sed.

For example, **ted\_UpChar** would have the following effect:

*Before:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

*After:*

I am the atomic powered robot.  
Please give my regards to  
everybody.

## Return Value

This routine returns TRUE if it succeeds in moving the cursor; otherwise, it returns FALSE.

## See Also

**ted\_SetMoveMethod**, **ted\_DownChar**, **ted\_LeftChar**, **ted\_RightChar**

## Note

This routine is implemented as a macro.

## Example

```
switch (scancode) {  
case UP:  
    ted_UpChar(sed);  
    break;  
  
/* ... */  
}
```

**Synopsis**

```
long ted_WriteFile(sed, fp, flag);  
    sed_type sed;      the sed  
    FILE *fp;          the file stream to write to  
    int flag           output style flag
```

**Description**

This routine writes the entire contents of the text buffer into file *fp*. It returns the number of characters written to the file.

You must open the file before calling **ted\_WriteFile**. Text files should be opened in text mode; binary files should be opened in binary mode.

*flag* determines how the text is written. It can have one of the following values:

- TED\_HARD** Write the text as it appears on the display with '\n' characters at the end of each line.
- TED\_SOFT** Write the text as it is stored in the text buffer with '\n' characters only where they actually exist.

**Return Value**

If successful this routine returns the number of characters written; otherwise, it returns -1L.

**See Also**

**ted\_ReadFile**

**Example**

```
FILE *fp;  
  
/* ... */  
case ALT_W:  
    fp = fopen(file_name, "w");  
    ted_WriteFile(sed, fp, TED_HARD);  
    fclose(fp);  
    break;
```

## Synopsis

```
#include <time.h>

#include "ctime.h"

struct tm *tm_AddDays(t, days);

    struct tm *t;           pointer to time structure
    long days;             number of days to add
```

## Description

The **tm\_AddDays** function adds an elapsed time in days to the **tm** struct to which *t* points and places the new adjusted time in the static result structure.

## Return Value

Returns a pointer to the result structure. The result struct is a static structure used by the time functions. Subsequent calls to time functions will destroy the contents of the result structure. If you need the result, copy it to another **tm** struct.

## See Also

**tm\_AddSecs**, **tm\_ElapDays**

## Example

```
struct tm now, then;

/* ... */

tm_Copy(&then, tm_AddDays(&now, 1000L));
```



### Synopsis

```
#include <time.h>

#include "ctime.h"

struct tm *tm_AddSecs(t, seconds);

    struct tm *t;           pointer to time structure
    long seconds;          number of seconds to add
```

### Description

The **tm\_AddSecs** function adds an elapsed time in seconds to the **tm** struct to which *t* points and places the new adjusted time in the static result structure.

### Return Value

Returns a pointer to the result structure. The result struct is a static structure used by the time functions. Subsequent calls to time functions will destroy the contents of the result structure. If you need the result, copy it to another **tm** struct.

### See Also

**tm\_AddDays**, **tm\_ElapSecs**

### Example

```
struct tm now, then;

/* ... */

tm_Copy(&then, tm_AddSecs(&now, 1000L));
```

### Synopsis

```
#include <time.h>

#include "ctime.h"

struct tm *tm_Adjust(t);

    struct tm *t;           pointer to time structure
```

### Description

This routine adjusts all the elements of the **tm** structure to which *t* points so that it contains a valid date. The routine starts with the smallest units (*tm\_sec*) and adjusts overflows until everything is correct. Then it calculates *tm\_wday* (day of the week), *tm\_yday* (day of the year), and sets the *tm\_isdst* (is daylight savings time) flag.

### Return Value

Returns a pointer to the modified structure, *t*.

### Example

```
struct tm date;
int dow;

/* get date using date_funcs*/
/* ... */
/* now find day of the week, dow */

tm_Adjust(&date);
dow = date.tm_wday;
```

**Synopsis**

```
#include <time.h>

#include "ctime.h"

int tm_Cmp(t1, t2);
    struct tm *t1;           pointer to first time structure
    struct tm *t2;           pointer to second time structure
```

**Description**

This routine compares two times and determines which is the earlier time.

**Return Value**

Returns a value indicating the relationship between the two times:

Less than 0	<i>t1</i> earlier than <i>t2</i> .
0	<i>t1</i> equivalent to <i>t2</i> .
Greater than 0	<i>t1</i> later than <i>t2</i> .

**Example**

```
struct tm now, then;

/* ... */

if (tm_Cmp(&now, &then) > 0) {
    /* ... */
}
```

## Synopsis

```
#include <time.h>

#include "ctime.h"

struct tm *tm_Copy(dest, source);

    struct tm *dest;      pointer to destination time structure
    struct tm *source;    pointer to source time structure
```

## Description

This routine copies the time in the *source* structure to the *dest* structure.

## Return Value

Returns *dest* or NULL if either *dest* or *source* is equal to NULL.

## See Also

**tm\_Zero**

## Example

```
struct tm now, then;

/* ... */

tm_Copy(&then, &now);
```

## Synopsis

```
#include <time.h>

#include "ctime.h"

int tm_DayOfWeek(mday, month, year);

    int mday;           day in month (0 through 31)
    int month;          month (0 through 11; January = 0)
    int year;           year (base year = 1900)
```

## Description

This routine calculates the day of the week for the given day in a given month.

## Return Value

Returns the week day number of the given day in *month, year*. The week day number is between 0 and 6 with 0 indicating Sunday.

## Example

```
/* ... */

/* Find first day of 1984 */
wday = tm_DayOfWeek(0, 0, 84);

/* ... */
```

## Synopsis

```
#include <time.h>

#include "ctime.h"

int tm_DaysInMonth(month, year);

    int month;           month (0 through 11; January = 0)
    int year;            year (base year = 1900)
```

## Description

This routine calculates the number of days in a given month.

## Return Value

Returns the number of days in *month*, *year*.

## Example

```
/* ... */

/* find number of days in August, 1974 */
ndays = tm_DaysInMonth(7, 74);

/* ... */
```

## tm\_ElapDays

Compute elapsed days between times

---

### Synopsis

```
#include <time.h>

#include "ctime.h"

long tm_ElapDays(t1, t2);

    struct tm *t1;           pointer to first time structure
    struct tm *t2;           pointer to second time structure
```

### Description

This routine computes the elapsed time in days between time *t1* and time *t2*.

### Return Value

Returns the elapsed time in days.

### See Also

**tm\_AddDays, tm\_ElapSecs**

### Example

```
struct tm now, then;
long days;

/* ... */

days = tm_ElapDays(&then, &now);
```

**Synopsis**

```
#include <time.h>

#include "ctime.h"

long tm_ElapSecs(t1, t2);

    struct tm *t1;           pointer to first time structure
    struct tm *t2;           pointer to second time structure
```

**Description**

This routine computes the elapsed time in seconds between time *t1* and time *t2*.

**Return Value**

Returns the elapsed time in seconds.

**See Also**

**tm\_AddDays, tm\_ElapSecs**

**Example**

```
struct tm now, then;
long seconds;

/* ... */

seconds = tm_ElapSecs(&then, &now);
```



**Synopsis**

```
#include <time.h>

#include "ctime.h"

struct tm *tm_Init(t);
    struct tm *t;           pointer to time structure
```

**Description**

This routine initializes the elements of the time structure to represent the time: midnight, December 31, 1969. The format used complies with the **Standard C** specification.

**Return Value**

Returns *t*.

**See Also**

**tm\_Now, ott\_Init, ott\_Now**

**Example**

```
struct tm t;

/* initialize time */
tm_Init(&t);
```

### Synopsis

```
#include <time.h>

#include "ctime.h"

boolean tm_IsDateValid(t);

    struct tm *t;           pointer to time structure
```

### Description

This routine determines if the date portion (*tm\_mday*, *tm\_mon*, and *tm\_year*) of the time structure is valid.

### Return Value

Returns TRUE if the date is valid, FALSE otherwise.

### See Also

**tm\_IsValid**

### Example

```
struct tm t;

/* ... */

if (tm_IsDateValid(&t)) {
    /* ... */
}
```

**Synopsis**

```
#include <time.h>

#include "ctime.h"

boolean tm_IsTimeValid(t);
    struct tm *t;           pointer to time structure
```

**Description**

This routine determines if the time portion (*tm\_sec*, *tm\_min*, and *tm\_hour*) of the time structure is valid.

**Return Value**

Returns TRUE if the time is valid, FALSE otherwise.

**See Also**

**tm\_IsDateValid, tm\_IsValid**

**Example**

```
struct tm t;

/* ... */

if (tm_IsTimeValid(&t)) {
    /* ... */
}
```

### Synopsis

```
#include <time.h>

#include "ctime.h"

boolean tm_IsValid(t);
    struct tm *t;           pointer to time structure
```

### Description

The **tm\_IsValid** function determines if the time structure to which *t* points is valid.

A valid date must meet the following criteria:

<i>tm_sec</i>	0 through 59.
<i>tm_min</i>	0 through 59.
<i>tm_hour</i>	0 through 23.
<i>tm_mday</i>	1 through 28, 29, 30, or 31 (varies according to month and year).
<i>tm_mon</i>	0 through 11.
<i>tm_year</i>	Greater than or equal to 0.
<i>tm_wday</i>	0 through 6.
<i>tm_yday</i>	0 through 364 or 365 (varies according to year).
<i>tm_isdst</i>	ignored.

### Return Value

Returns TRUE if the time is valid, FALSE otherwise.

### See Also

**tm\_IsDateValid, tm\_IsTimeValid**

### Example

```
struct tm t;

if (tm_IsValid(&t)) {    /* ... */
```

**Synopsis**

```
#include <time.h>

#include "ctime.h"

struct tm *tm_Now(t);
    struct tm *t;           pointer to time structure
```

**Description**

This routine sets all of the elements of a time structure to the current time. The format used complies with the **Standard C** specification.

**Return Value**

Returns *t*.

**See Also**

**ott\_Now, tm\_Zero, tm\_Init, ott\_Init**

**Example**

```
struct tm t;

/* get the current time */
tm_Now(&t);
```

### Synopsis

```
#include <time.h>

#include "ctime.h"

TIME_T tm_StructToTt(t, tt);

    struct tm *t;           pointer to the (source) time structure
    TIME_T *tt;             pointer to TIME_T (destination)
```

### Description

This routine converts the time/date represented by the members of the given time structure into a **TIME\_T** representation and returns this value. If *tt* is not NULL, the converted value is also stored at this address.

### Return Value

Returns the time/date of the given tm struct encoded as a TIME\_T type.

### See Also

**tm\_TtToStruct**

### Note

The format used to interpret the value of the **TIME\_T** variable is system dependent. If the system's compiler is Standard C compliant, the value is interpreted using the system's native format. In this case, you are free to mix OWL function calls, C-scape field functions, and calls to your system's libraries (like using **time()** to initialize a field function's variable), since all functions use the same format for TIME\_T variables. On other systems, the value is interpreted in accordance with the POSIX standard. The library functions on these systems may use a different format to interpret TIME\_T values. This would make OWL functions and C-scape field functions incompatible with system library functions. By always restricting yourself to only OWL library functions (like using **ott\_Now()** in place of **time()**), you will maintain the highest level of portability whether your target system is Standard C compliant, or not.

## Example

```
struct tm t;  
TIME_T tt;  
  
/* put the current time in TIME_T variable tt */  
tm_StructToTt(tm_Now(&t), &tt); /* equivalent to ott_Now(&tt) */
```

**Synopsis**

```
#include <time.h>

#include "ctime.h"

struct tm *tm_TtToStruct(tt, t);

    TIME_T tt;           the (source) TIME_T variable
    struct tm *t;        pointer to the (destination) time structure
```

**Description**

This routine converts the time/date represented by the given TIME\_T argument, *tt*, into a time structure format and sets all of the elements of the given time structure, *t*, accordingly. The format used complies with the Standard C specification.

**Return Value**

Returns *t*.

**See Also**

**tm\_StructToTt**

**Note**

The format used to store the value of the TIME\_T variable is system dependent. If the system's compiler is Standard C compliant, the value is stored using the systems native format. In this case, you are free to mix OWL function calls, C-scape field functions, and calls to your system's libraries (like using **time()** to initialize a field function's variable), since all functions use the same format for TIME\_T variables. On other systems, the value is stored in accordance with the POSIX standard. The library functions on these systems may use a different format to store values in TIME\_T variables which would make OWL functions and C-scape field functions incompatible with system library functions. By always restricting yourself to only OWL library functions (like using **ott\_Now()** in place of **time()**), you will maintain the highest level of portability whether your target system is Standard C compliant, or not.



## Example

```
struct tm t;  
  
/* put the current time in TIME_T variable tt */  
tm_TtToStruct(ott_Now(NULL), &t); /* equivalent to tm_Now(&t) */
```

**Synopsis**

```
#include <time.h>

#include "ctime.h"

struct tm *tm_Zero(t);
    struct tm *t;           pointer to time structure
```

**Description**

This routine sets all the elements of the time structure to 0.

**Return Value**

Returns *t*.

**See Also**

**tm\_Copy**

**Example**

```
struct tm t;

/* initialize time */
tm_Zero(&t);
```

**Synopsis**

```
boolean valid_Double(num, string);  
    double num;           the value to be checked  
    char *string;         the validation string
```

**Description**

This routine parses a validation string for a numeric range specification and then determines if a value is within the specified range. It is used by field functions that have a double variable type.

The validation string consists of a series of number pairs. Each number pair is enclosed in parentheses with the numbers separated by a comma. Each number pair specifies a lower and upper range. As many number pairs as desired can be used. A value is within range if it falls within the lower and upper bounds of at least one of the number pairs. The validation string “(10.2,22)” specifies a range of 10.2 through 22, inclusive. The validation string “(2,5.4)(8.6,15.2)” specifies a range of 2 through 5.4 or 8.6 through 15.2. A missing number, such as “(4.7,)”, indicates an open-ended range. In the preceding case a value greater than or equal to 4.7 would be valid.

The validation strings are not error-checked so make sure they are properly formed.

The numeric field **fexit** functions check the second field data pointer (data pointer number 1) for a validation string. If one is present then the appropriate validation function is called. If the value in the field is in range the **fexit** function returns TRUE otherwise an out-of-range message is sent to the border prompt (if a border is attached) and FALSE is returned.

**Return Value**

Returns TRUE if *num* is within the range specified by the format string. Returns FALSE otherwise.

**See Also**

**valid\_Float, valid\_Int, valid\_Long, valid\_String**

## Example

```
boolean my_double_fexit(sed)
    sed_type sed;
/*
    Make sure the number entered is between 0 and 99.5.
*/
{
    double val;

    sscanf(sed_GetCurrRecord(sed), "%lf", &val);

    if ( !valid_Double(val, "(0,99.5)" ) ) {
        tone();
        sed_BorderPrompt(sed, "Number out of range");

        /* wait for a keystroke */
        while (!kb_Check())
            ;
        sed_BorderPrompt(sed, NULL);
        return(FALSE);
    }

    return(TRUE);
}
```

### Synopsis

```
boolean valid_Float(num, string);  
    float num;           the value to be checked  
    char *string;        the validation string
```

### Description

This routine parses a validation string for a numeric range specification and then determines if a value is within the specified range. It is used by field functions that have a float variable type.

The validation string consists of a series of number pairs. Each number pair is enclosed in parentheses with the numbers separated by a comma. Each number pair specifies a lower and upper range. As many number pairs as desired can be used. A value is within range if it falls within the lower and upper bounds of at least one of the number pairs. The validation string “(10.2,22)” specifies a range of 10.2 through 22, inclusive. The validation string “(2,5.4)(8.6,15.2)” specifies a range of 2 through 5.4 or 8.6 through 15.2. A missing number, such as “(4.7,)”, indicates an open-ended range. In the preceding case a value greater than or equal to 4.7 would be valid.

The validation strings are not error-checked so make sure they are properly formed.

The numeric field **fxit** functions check the second field data pointer (data pointer number 1) for a validation string. If one is present then the appropriate validation function is called. If the value in the field is in range the **fxit** function returns TRUE otherwise an out-of-range message is sent to the border prompt (if a border is attached) and FALSE is returned.

### Return Value

Returns TRUE if *num* is within the range specified by the format string. Returns FALSE otherwise.

### See Also

**valid\_Double, valid\_Int, valid\_Long, valid\_String**

## Note

This routine is implemented as a macro.

## Example

```
boolean my_float_fexit(sed)
    sed_type sed;
/*
    Make sure the number entered is between 0 and 99.5.
*/
{
    float val;

    sscanf(sed_GetCurrRecord(sed), "%f", &val);

    if ( !valid_Float(val, "(0,99.5)" ) ) {
        tone();
        sed_BorderPrompt(sed, "Number out of range");

        /* wait for a keystroke */
        while (!kb_Check())
            ;
        sed_BorderPrompt(sed, NULL);
        return(FALSE);
    }

    return(TRUE);
}
```

**Synopsis**

```
boolean valid_Int(num, string);  
  
    int num;           the value to be checked  
    char *string;      the validation string
```

**Description**

This routine parses a validation string for a numeric range specification and then determines if a value is within the specified range. It is used by field functions that have a `int` variable type.

The validation string consists of a series of number pairs. Each number pair is enclosed in parentheses with the numbers separated by a comma. Each number pair specifies a lower and upper range. As many number pairs as desired can be used. A value is within range if it falls within the lower and upper bounds of at least one of the number pairs. The validation string "(0,22)" specifies a range of 0 through 22, inclusive. The validation string "(2,5)(44,55)" specifies a range of 2 through 5 or 44 through 55. A missing number, such as "(4,)", indicates an open-ended range. In the preceding case a value greater than or equal to 4 would be valid.

The validation strings are not error-checked so make sure they are properly formed.

The numeric field **fexit** functions check the second field data pointer (data pointer number 1) for a validation string. If one is present then the appropriate validation function is called. If the value in the field is in range the **fexit** function returns **TRUE** otherwise an out-of-range message is sent to the border prompt (if a border is attached) and **FALSE** is returned.

**Return Value**

Returns **TRUE** if *num* is within the range specified by the format string. Returns **FALSE** otherwise.

**See Also**

**valid\_Double, valid\_Float, valid\_Long, valid\_String**

## Note

This routine is implemented as a macro.

## Example

```
boolean my_int_fexit(sed)
    sed_type sed;
/*
    Make sure the number entered is between 0 and 100.
*/
{
    int val;

    sscanf(sed_GetCurrRecord(sed), "%d", &val);

    if ( !valid_Int(val, "(0,100)" ) ) {
        tone();
        sed_BorderPrompt(sed, "Number out of range");

        /* wait for a keystroke */
        while (!kb_Check())
            ;
        sed_BorderPrompt(sed, NULL);
        return(FALSE);
    }

    return(TRUE);
}
```



**Synopsis**

```
boolean valid_Long(num, string);  
    long num;           the value to be checked  
    char *string;       the validation string
```

**Description**

This routine parses a validation string for a numeric range specification and then determines if a value is within the specified range. It is used by field functions that have a long variable type.

The validation string consists of a series of number pairs. Each number pair is enclosed in parentheses with the numbers separated by a comma. Each number pair specifies a lower and upper range. As many number pairs as desired can be used. A value is within range if it falls within the lower and upper bounds of at least one of the number pairs. The validation string "(0,22)" specifies a range of 0 through 22, inclusive. The validation string "(2,5)(44,55)" specifies a range of 2 through 5 or 44 through 55. A missing number, such as "(4,)", indicates an open-ended range. In the preceding case a value greater than or equal to 4 would be valid.

The validation strings are not error-checked so make sure they are properly formed.

The numeric field **fexit** functions check the second field data pointer (data pointer number 1) for a validation string. If one is present then the appropriate validation function is called. If the value in the field is in range the **fexit** function returns TRUE otherwise an out-of-range message is sent to the border prompt (if a border is attached) and FALSE is returned.

**Return Value**

Returns TRUE if *num* is within the range specified by the format string. Returns FALSE otherwise.

**See Also**

**valid\_Double, valid\_Float, valid\_Int, valid\_String**

## Note

This routine is implemented as a macro.

## Example

```
boolean my_long_fexit(sed)
    sed_type sed;
/*
    Make sure the number entered is between 0 and 100.
*/
{
    long val;

    sscanf(sed_GetCurrRecord(sed), "%ld", &val);

    if ( !valid_Long(val, "(0,100)" ) ) {
        tone();
        sed_BorderPrompt(sed, "Number out of range");

        /* wait for a keystroke */
        while (!kb_Check())
            ;
        sed_BorderPrompt(sed, NULL);
        return(FALSE);
    }

    return(TRUE);
}
```

### Synopsis

```
boolean valid_String(string, vstr);  
    char *string;           the string to be checked  
    char *vstr;            the validation string
```

### Description

This routine parses a validation string for a list of allowed string values and determines if a string is within the specified list. It is used by field functions that have char \* variable type.

The validation string consists of a series of string values. Each string value is separated by a comma. As many strings as desired can be listed. A string is valid if it is equal to one of the entries in the list.

The validation strings are not error-checked so make sure they are properly formed. Make sure the case of the string values is correct.

The string field **fexit** functions check the second field data pointer (data pointer number 1) for a validation string. If one is present then the appropriate validation function is called. If the value in the field is in range the **fexit** function returns TRUE otherwise an error message is sent to the border prompt (if a border is attached) and FALSE is returned.

### Return Value

Returns TRUE if *string* is in the list of strings range specified by the validation string. Returns FALSE otherwise.

### See Also

**valid\_Double, valid\_Float, valid\_Int, valid\_Long**

## Example

```
#define WDAY_LIST  "Monday,Tuesday,Wednesday,Thursday,Friday,Sa-  
turday,Sunday"  
  
boolean day_fexit(sed)  
    sed_type sed;  
/*  
    Make sure the string entered is a valid week day.  
*/  
{  
    if ( !valid_String(sed_GetCurrRecord(sed), WDAY_LIST)) {  
        tone();  
        sed_BorderPrompt(sed, "Not a week day name");  
  
        /* wait for a keystroke */  
        while (!kb_Check())  
            ;  
        sed_BorderPrompt(sed, NULL);  
        return(FALSE);  
    }  
  
    return(std_fexit(sed));  
}
```

## Appendix B: Field Function Reference

---

## Preface

This appendix is a reference section for the various field functions provided with C-scape. Each page layout is of the form below, discussing the purpose and implementation specifics for each set of field functions. C-scape's flexibility for creating custom data entry screens stems mostly from the field function structure concept. The field functions described here were designed to handle a wide variety of data entry needs, but by no means is C-scape limited to only these types of fields. Please notice that many field function structures described here contain component functions that are shared. The same components can be combined in countless ways to create any type of field.

## Description

A brief explanation of what the field does and how it does it.

## Variable Type

The C language variable type that the field expects. In all cases this is a **pointer**.

## Field Data Pointers

Most field functions use the first three field data pointers in the standard way listed below. These data pointers are always optional; if you don't need them set them to NULL or simply leave them unspecified. Any other expectations by the field functions will be explained individually.

- 0) Prompt message
- 1) Validation String
- 2) Format String

## Field Function Structure

The actual **field\_funcs\_struct** initialization.

## Source File

The name of the file that contains the source code.

## See Also

Other functions that behave in a similar manner or operate on a similar data type.

## Example

All of the examples in this reference use the code fragment below. Only the variables specific to the example and the **menu\_Printf** calls were included in each reference.

```

#include "cscape.h"

void main()
{
    menu_type menu;
    sed_type sed;

    disp_Init(def_ModeText, FNULL);    /* Initialize display */

    menu = menu_Open();                /* Create the menu object */

    menu_Printf(menu, ...);            /* Define the menu,
                                        from the example */

    menu_Flush(menu);                 /* Finish defining the menu */

    sed = sed_Open(menu);              /* Create the sed object */

    sed_SetPosition(sed, 10, 10);      /* Customize the sed */




    sed_Repaint(sed);                 /* Display the sed */
    sed_Go(sed);                      /* Execute the sed */

    sed_Close(sed);                   /* Destroy the sed(and menu)*/

    disp_Close();                     /* Finish with the display */
}

```

## Description

Allows the entry of only non-numeric standard ASCII characters (a - z, A - Z). The  and  keys move through the field and the  key toggles insert/overwrite mode. The cursor is enlarged in insert mode.

## Variable Type

char \*

## Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

## Field Function Structure

```
field_funcs_struct alpha_funcs = {  
    stdBigCur_fenter,  
    string_fexit,  
    alpha_fkey,  
    string_senter,  
    string_sexit,  
    VAR_STRING  
};
```

## Source File

*fnalpha.c*

## See Also

**string\_funcs, xstring\_funcs, char\_funcs**

## Example

```
char dinner[50];  
dinner[0] = '\0';  
  
menu_Printf(menu, "What are we having for dinner ? @fd2[#####]",  
    dinner, &alpha_funcs, "(Hunan, Italian)", ",Hunan,Italian");
```



### Description

Passes control to a bob attached at a field. When actions occur in an embedded bob, **bob\_funcs** makes them appear to occur in the outer sed. **bob\_fkey** merely calls **bob\_Go** then checks for the return value for field movement codes (BOB\_UP, BOB\_DOWN, BOB\_LEFT, BOB\_RIGHT, BOB\_QUIT).

### Variable Type

VOID

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct bob_funcs = {
    stdNoCur_fenter,
    std_fexit,
    bob_fkey,
    FNULL,
    FNULL,
    0
};
```

### Source File

*fnbob.c*

### See Also

**bobint\_funcs**, **sled\_funcs**

### Example

```
sed_type ted;
char note[100];
note[0] = '\0';

menu = menu_Open();
menu_Printf(menu, "@f[]", note, &ted_funcs);
ted = sed_Open(menu);
ted_SetMaxSize(ted, 100L);

menu_Printf(menu, "@fbd[]", NULL, &bob_funcs,
    sed_CreateBob(ted, BOB_DEPENDENT), "A small note pad.");
```

### Description

Passes control to a bob attached at a field. When actions occur in a bob, **bob\_funcs** makes them appear to occur in the outer sed. **bob\_fkey** merely calls **bob\_Go** then checks for the return value for field movement codes (BOB\_UP, BOB\_DOWN, BOB\_LEFT, BOB\_RIGHT, BOB\_QUIT).

When used in conjunction with **radiobob\_funcs** (see below), the variable holds the number the selected field within the bob.

### Variable Type

int\*

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct bobint_funcs = {
    stdNoCur_fenter,
    std_fexit,
    bob_fkey,
    FNULL,
    FNULL,
    sizeof(int*)
};
```

### Source File

*fnbobint.c*

### See Also

**bob\_funcs**

## Example

```
menu_type  menu, radiomenu;
sed_type   sed, radiosed;
int        top_stooge = 0; /* No default (must initialize this)
*/
char       *stooge_name[4] = {"Moe", "Larry", "Curly", "Shemp"};
/* ... */
/* create the radio button bob */
radiomenu = menu_Open();
menu_Printf(radiomenu, "@f[ # Moe  ]\n", NULL,
            &radiobob_funcs);
menu_Printf(radiomenu, "@f[ # Larry ]\n", NULL,
            &radiobob_funcs);
menu_Printf(radiomenu, "@f[ # Curly ]\n", NULL,
            &radiobob_funcs);
menu_Printf(radiomenu, "@f[ # Shemp ]", NULL,
            &radiobob_funcs);
menu_Flush(radiomenu);
radiosed = sed_Open(radiomenu);
sed_SetSpecial(radiosed, spc_Embed);
/* ... */
/* create the parent sed */
menu = menu_Open();
menu_Printf(menu, "Favorite stooge:\n\n");
menu_Printf(menu, "@fbd[]", &top_stooge, &bobint_funcs,
            sed_CreateBob(radiosed, BOB_DEPENDENT), "Choose a stooge");
/* ... after disp_Close */
if (top_stooge != 0) {
    printf("\n%s is top stooge!\n", stooge_name[top_stooge - 1]);
}
else {
    printf("\nNo choice was made\n");
}
```

### Description

Allows the entry of a single character. The field should have only one writeable position. The variable is a `char *`, but not a string; no terminating `'\0'` character is written to the variable.

### Variable Type

`char *`

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct char_funcs = {  
    std_fenter,  
    string_fexit,  
    char_fkey,  
    char_senter,  
    char_sexit,  
    sizeof(char)  
};
```

### Source File

*fnchar.c*

### See Also

[string\\_funcs](#), [xstring\\_funcs](#), [alpha\\_funcs](#)

### Example

```
char digit = '0';  
  
menu_Printf(menu, "Select a digit : @fd2[#]", &digit,  
    &char_funcs, "0 - 9", "0,1,2,3,4,5,6,7,8,9");
```

## check\_funcs

Make multiple selections with a check mark

---

### Description

Allows the selection of more than one choice from a list of several options. The field spec should contain the text representing the choice and a single writeable position. A check mark is toggled in the writeable position when the Space Bar is pressed. The field's variable is a boolean that is TRUE if the field was selected and FALSE if not.

### Variable Type

boolean \*

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct check_funcs = {
    stdNoCur_fenter,
    std_fexit,
    check_fkey,
    check_senter,
    check_sexit,
    sizeof(boolean)
};
```

### Source File

*fncheck.c*

### See Also


**mark\_funcs, yesno\_funcs**

### Example

```
boolean choice[4];

menu_Printf(menu, "@f[# Bread ]", &choice[0], &check_funcs);
menu_Printf(menu, "@f[# Butter]", &choice[1], &check_funcs);
menu_Printf(menu, "@f[# Eggs  ]", &choice[2], &check_funcs);
menu_Printf(menu, "@f[# Cream ]", &choice[3], &check_funcs);
```

### Description

Treats the field as a menu choice. Supports grid movement among fields. Pressing a field's first letter or  makes a selection and exits the sed.

### Variable Type

VOID

### Field Data Pointers

- 0) Prompt message
- 1) Return value string

### Field Function Structure

```
field_funcs_struct click_funcs = {  
    stdNoCur_fenter,  
    std_fexit,  
    click_fkey,  
    FNULL,  
    FNULL,  
    0  
};
```

### Source File

*fnmenu.c*

### See Also

**menu\_funcs, gmenu\_funcs**

## Example

```
/* from demomous.c */
/* ... */
menuone = menu_Open();
menu_Printf(menuone, " Name:      @fd[#####] \n\n",
    name, &string_funcs, "Enter customer name");
menu_Printf(menuone, " City:      @fd[#####] \n\n",
    city, &alpha_funcs, "Enter customer city");
menu_Printf(menuone, " State:      @fd2[##      ] \n\n", state,
    &alpha_funcs,
    "Enter state abbrev.", STATE_2NAMES);
menu_Printf(menuone, "           @fd[ Done ]",
    NULL, &click_funcs, "Click to quit");

/* ... */
```

### Description

Allows calculator style entry of a long integer. It displays commas automatically and reserves the left most position in the field's record for a negative sign. It adds digits at the right and pushes others to the left. The  and  keys delete the right most digit and pull the rest to the right. The  clears the field.

This functions saves enough space at the left of the field to display a minus sign, which users can toggle with the  key.

### Variable Type

long \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct clong_funcs = {
    snum_fenter,
    clong_fexit,
    clong_fkey,
    clong_senter,
    clong_sexit,
    sizeof(long)
};
```

### Source File

*fnclong.c*

### See Also

**cmoney\_funcs**, **long\_funcs**, **plong\_funcs**, **slong\_funcs**



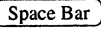
### Example


```
long value = 0L;

menu_Printf(menu, "Value : @fd[#####]",
    &value, &clong_funcs, "Enter a big value");
```



**Description**

Allows calculator style entry of a long integer representing 1/100's of a unit of currency, forcing the last two digits behind a fixed decimal point. It displays commas automatically and reserves the left most position of the field's record for a negative sign. It adds digits at the right and pushes others to the left. The  and  keys delete the right most digit and pull the rest to the right. The  clears the field.

By default, this function saves enough space at the left of the field to display a minus sign, which users can toggle with the  key. If you wish to disable this feature, undefine the symbol MINUS and recompile this file.

**Variable Type**

long \*

**Field Data Pointers**

- 0) Prompt message
- 1) Validation string
- 2) Format string

**Field Function Structure**

```
field_funcs_struct cmoney_funcs = {
#ifdef MINUS
    snum_fenter,
#else
    num_fenter,
#endif
    cmoney_fexit,
    cmoney_fkey,
    cmoney_senter,
    cmoney_sexit,
    sizeof(long)
};
```

**Source File**

*fncmoney.c*

## **See Also**

**money\_funcs**

## **Example**

```
long salary = 0L;

menu_Printf(menu, "Salary : @fd[$ #####]",
    &salary, &money_funcs, "Your yearly income.");
```

### Description

Allows the entry of a date in the form MM/DD/YY. The field should have only six writeable positions. It places the results in the date portion of the **tm struct**. Refer to the chapter “Higher Level Functions” in the *C-scape Manual* for information about **tm structs**.

You can change the date format used by **date\_funcs** by changing the *date\_fmt* element of the **ocountry\_struct**. Its values can include MMDDYY, DDMMYY, and YYMMDD.

```
ocountry.date_fmt = DDMMYY;
```

Refer to the *C-scape Manual* for more on the **ocountry\_struct**.

**#define OAK\_PRE40TM** for tm struct backward compatibility with prior versions.

### Variable Type

```
struct tm *
```

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct date_funcs = {
    std_fenter,
    date_fexit,
    date_fkey,
    date_senter,
    date_sexit,
    sizeof(struct tm)
};
```

### Source File

*fndate.c*

### See Also

**datet\_funcs**, **time\_funcs**, **timet\_funcs**

## Example

```
#include "ctime.h"          /* necessary for the time functions */

struct tm date;

tm_Now(&date);              /* initialize the date */

menu_Printf(menu, "Date : @fd[##/##/##]", &date, &date_funcs,
    "Enter your birthday in the form month/day/year.");
```

### Description

Allows the entry of a date in the form MM/DD/YY. The field should have only six writeable positions. It places the results in a `TIME_T` variable. Refer to the chapter “Higher Level Functions” in the *C-scape Manual* for information about C-scape time functions.

You can change the date format used by **datet\_funcs** by changing the *date\_fmt* element of the **ocountry\_struct**. Its values can include MMDDYY, DDMMYY, and YYMMDD.

```
ocountry.date_fmt = DDMMYY;
```

Refer to the *C-scape Manual* for more on the **ocountry\_struct**.

### Variable Type

`TIME_T *`

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct datet_funcs = {
    std_fenter,
    datet_fexit,
    date_fkey,
    datet_senter,
    datet_sexit,
    sizeof(TIME_T)
};
```

### Source File

*fndatet.c*

### See Also

**date\_funcs**, **time\_funcs**, **timet\_funcs**

## Example

```
#include "ctime.h"    /* necessary for the time functions */

TIME_T date;

ott_Now(&date);       /* initialize the date */

menu_Printf(menu, "Date : @fd[##/##/##]", &date, &datet_funcs,
    "Enter your birthday in the form month/day/year.");
```

### Description

Allows entry of a string consisting of only numeric characters. The **→** and **←** keys move through the field and the **Ins** key toggles insert/overwrite mode.

### Variable Type

char \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct digit_funcs = {  
    stdBigCur_fenter,  
    string_fexit,  
    digit_fkey,  
    string_senter,  
    string_sexit,  
    VAR_STRING  
};
```

### Source File

*fndigit.c*



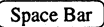
### See Also

**string\_funcs**, **xstring\_funcs**, **char\_funcs**, **alpha\_funcs**

### Example

```
char age[10];  
  
age[0] = '\0';  
  
menu_Printf(menu, "Age @fd[####]", age,  
    &digit_funcs, "Enter your age");
```

### Description

Allows scientific calculator style input of a double value. The field should contain at least eight writeable positions and no non-writeables. The field always contains an 'e' character to separate the mantissa and exponent. Space for a negative sign and three digits are always reserved in the exponent area; e.g., a field with eight writeables containing the value 0.01234 would appear as : 1.2e-004. Pressing the 'e' key moves the user from the mantissa area to the exponent area, and vice versa. **double\_funcs** adds digits at the right and pushes others to the left. The  and  keys delete the right most digit of the appropriate area and pull the rest to the right. The  clears the field.

### Variable Type

double \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct double_funcs = {
    double_fenter,
    double_fexit,
    double_fkey,
    double_senter,
    double_sexit,
    sizeof(double)
};
```

### Source File

*fndouble.c*

### See Also

**sdouble\_funcs**, **sfloat\_funcs**




## Example

```
double value = 2.0;
```

```
menu_Printf(menu, "What is the square root of three? @fd[22,#]",  
    &value, &double_funcs, "2 is not close enough!");
```

### Description

Controls all the behavior that framer (pulldown) menus exhibit. It checks each field for a bob; if the bob is a sed, **framer\_funcs** displays it and passes control. If the bob is a user function **framer\_funcs** passes control when the  key is pressed. If the field has no bob, then the integer associated with that field (assumed to be contained in a string attached to the field's second generic data pointer) is returned through the original **frame\_Go** call.

### Variable Type

VOID

### Field Data Pointers

- 0) Prompt message
- 1) Integer associated with choice

### Field Function Structure

```
field_funcs_struct framer_funcs = {  
    stdNoCur_fenter,  
    std_fexit,  
    framer_fkey,  
    FNULL,  
    FNULL,  
    0  
};
```

### Source File

*fnframer.c*

### See Also

**slug\_funcs**

### Description

Allows the selection of a field as a menu choice. The field should contain no writeable positions. Field movement uses the grid. The `[Home]` and `[End]` keys move to the first and last fields of the sed respectively. Entering any alphanumeric character invokes a search for the first field beginning with that character.

Pressing the `[←]` key exits the sed and places the current field number plus one in the baton. You can override this, however, by putting a numeric string, such as “22”, in the field’s second data pointer (data pointer 1). **gmenu\_funcs** converts this string into an integer and places it in the baton when the `[←]` key is pressed.

### Variable Type

VOID

### Field Data Pointers

- 0) Prompt message
- 1) Return value string

### Field Function Structure

```
field_funcs_struct gmenu_funcs = {
    stdNoCur_fenter,
    std_fexit,
    gmenu_fkey,
    FNULL,
    FNULL,
    0
};
```

### Source File

*fnmenu.c*



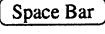
### See Also

**menu\_funcs**

## Example

```
menu_Printf(menu, "@p[0,0]@f[EENIE ]", NULL, &gmenu_funcs);  
menu_Printf(menu, "@p[0,10]@f[MEENIE]", NULL, &gmenu_funcs);  
menu_Printf(menu, "@p[1,0]@f[MINEY ]", NULL, &gmenu_funcs);  
menu_Printf(menu, "@p[1,10]@f[MOE   ]", NULL, &gmenu_funcs);
```

### Description

Allows calculator style entry of an integer in hexadecimal format. It adds digits at the right and pushes others to the left. The  and  keys delete the right most digit and pull the rest to the right. The  clears the field.

### Variable Type

int \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct hex_funcs = {  
    num_fenter,  
    hex_fexit,  
    hex_fkey,  
    hex_senter,  
    hex_sexit,  
    sizeof(int)  
};
```

### Source File

*fnhex.c*



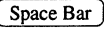
### See Also

**int\_funcs**

### Example

```
int hex = 0x0000;  
  
menu_Printf(menu, "Value : @fd[####]", &hex, &hex_funcs,  
    "Enter a hex value");
```

### Description

Allows calculator style entry of an integer. It reserves the left most position in the field's record for a negative sign. It adds digits at the right and pushes others to the left. The  and  keys delete the right most digit and pull the rest to the right. The  clears the field.

### Variable Type

int \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct int_funcs = {  
    snum_fenter,  
    int_fexit,  
    num_fkey,  
    int_senter,  
    int_sexit,  
    sizeof(int)  
};
```

### Source File

*fnint.c*



### See Also

[sint\\_funcs](#), [pint\\_funcs](#), [range\\_funcs](#), [digit\\_funcs](#)

### Example

```
int x = 0;  
  
menu_Printf(menu, "How many? : @fd2[#####]", &x, &int_funcs,  
    "Enter the number of them (0 - 15)", "(0,15)");
```

### Description

Presents a popup list of string choices when any key not used for field movement is pressed. The choices should be placed in a string separated by commas and attached to the field's second generic data pointer. The  key selects a choice and copies it into the field's variable. The field's variable should be large enough to contain the string and its terminating '\0' character. The  key aborts the popup.

### Variable Type

char \*

### Field Data Pointers

- 0) Prompt message
- 1) Choice list string

### Field Function Structure

```
field_funcs_struct list_funcs = {  
    stdNoCur_fenter,  
    std_fexit,  
    list_fkey,  
    string_senter,  
    string_sexit,  
    VAR_STRING  
};
```

### Source File

*fnlist.c*

### See Also

**toggle\_funcs, togint\_funcs, radio\_funcs, radiobob\_funcs**

### Example

```
char choice[50];  
  
choice[0] = '\0';  
  
menu_Printf(menu, "The programmer's best friend is @fd2[12,#]",  
    choice, &list_funcs,  
    "Any key for popup", "Coke,Jolt,Coffee,Dos Equis");
```

**Description**

Allows the calculator style entry of a long integer. It reserves the left most position in the field's record for a negative sign. It adds digits at the right and pushes others to the left. The **←** and **Del** keys delete the right most digit and pull the rest to the right. The **Space Bar** clears the field.

**Variable Type**

long \*

**Field Data Pointers**

- 0) Prompt message
- 1) Validation string
- 2) Format string

**Field Function Structure**

```
field_funcs_struct long_funcs = {
    snum_fenter,
    long_fexit,
    num_fkey,
    long_senter,
    long_sexit,
    sizeof(long)
};
```

**Source File**

*fnlong.c*

**See Also**

**clong\_funcs, slong\_funcs, money\_funcs, cmoney\_funcs**

**Example**

```
long value = 0L;

menu_Printf(menu, "Value : @fd2[#####]",
    &value, &long_funcs,
    "Enter a value 0 to 1500000", "(0,1500000)");
```



### Description

Allows the selection of more than one choice from a list of several options. The field spec should contain the text representing the choice. The field's highlighting is toggled when the `Space Bar` is pressed. The field's variable is a boolean that is TRUE if the field was selected and FALSE if not.

### Variable Type

boolean \*

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct mark_funcs = {
    stdNoCur_fenter,
    std_fexit,
    mark_fkey,
    mark_senter,
    mark_sexit,
    sizeof(boolean)
};
```

### Source File

*fnmark.c*

### See Also

**check\_funcs**, **yesno\_funcs**

### Example

```
boolean choice[4];

menu_Printf(menu, "@f[Bread ]", &choice[0], &mark_funcs);
menu_Printf(menu, "@f[Butter]", &choice[1], &mark_funcs);
menu_Printf(menu, "@f[Eggs  ]", &choice[2], &mark_funcs);
menu_Printf(menu, "@f[Cream ]", &choice[3], &mark_funcs);
```

### Description

Allows the selection of a field as a menu choice. The field should contain no writeable positions. Movement between fields is sequential (see **inter\_field**). The **Home** and **End** keys move to the first and last fields of the sed respectively. Entering any alphanumeric character invokes a search for the first field record beginning with that character.

Pressing the **↵** key exits the sed and places the current field number plus one in the baton. You can override this, however, by putting a numeric string, such as "22", in the field's second data pointer (data pointer 1). **menu\_funcs** converts this string into an integer and places it in the baton when the **↵** key is pressed.

### Variable Type

VOID

### Field Data Pointers

- 0) Prompt message
- 1) Return value string

### Field Function Structure

```
field_funcs_struct menu_funcs = {
    stdNoCur_fenter,
    std_fexit,
    menu_fkey,
    FNULL,
    FNULL,
    0
};
```

### Source File

*fnmenu.c*




### See Also


**click\_funcs**, **gmenu\_funcs**

### Example

```
menu_Printf(menu, "@f[Moe]\n", NULL, &menu_funcs);
menu_Printf(menu, "@f[Larry]\n", NULL, &menu_funcs);
menu_Printf(menu, "@f[Curly]", NULL, &menu_funcs);
```

### Description

Allows the calculator style entry of a long integer representing 1/100's of a unit of currency, and forces the last two digits behind a fixed decimal point. It displays commas automatically and reserves the left most position of the field's record for a negative sign. It adds digits at the right and pushes others to the left. The  and  keys delete the right most digit and pull the rest to the right. The  clears the field.

By default, this function saves enough space at the left of the field to display a minus sign, which users can toggle with the  key. If you wish to disable this feature, undefine the symbol MINUS and recompile this file.

The field must have at least five writeables (-0.00). If you undefine MINUS, it must have at least four.

### Variable Type

long \*

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct money_funcs = {
#ifdef MINUS
    snum_fenter,
#else
    num_fenter,
#endif
    money_fexit,
    money_fkey,
    money_senter,
    money_sexit,
    sizeof(long)
};
```

### Source File

*fnmoney.c*

### See Also

**cmoney\_funcs**, **long\_funcs**, **slong\_funcs**

## Example

```
long salary = 0L;  
  
menu_Printf(menu, "Salary : @fd[$ #####]",  
            &salary, &money_funcs, "Your yearly income.");
```

**Description**

Treats the field as a menu choice with grid movement among fields. It behaves identically to **menu\_funcs**, except that it uses **string\_senter**. The text for the field's menu selection is determined by the contents of the field's string variable. Hence, it can be changed at runtime, though the user may not edit the field directly. Upon exit, the value of the field is *not* written back to the variable.

**Variable Type**

char \*

**Field Data Pointers**

- 0) Prompt message
- 1) Return value string

**Field Function Structure**

```
field_funcs_struct nowrite_funcs = {
    stdNoCur_fenter,
    std_fexit,
    menu_fkey,
    string_senter,
    FNULL,
    VAR_STRING
};
```

**Source File**

*fnowrit.c*

**See Also**



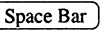
**menu\_funcs**, **gmenu\_funcs**

**Example**

```
char *stooge[3] = {"Moe", "Larry", "Curly"};
int i;

for (i = 0; i < 3; i++) {
    menu_Printf(menu, "@f[#####]\n", stooge[i], &nowrite_funcs);
}
```

### Description

Allows the calculator style entry of a non-negative integer. It adds digits at the right and pushes others to the left. The  and  keys delete the right most digit and pull the rest to the right. The  clears the field. The variable for **pint\_funcs** is an int *not* an unsigned int.

### Variable Type

int \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct pint_funcs = {  
    num_fenter,  
    int_fexit,  
    pnum_fkey,  
    int_senter,  
    int_sexit,  
    sizeof(int)  
};
```

### Source File

*fnpnum.c*




### See Also

**int\_funcs, sint\_funcs, digit\_funcs**

### Example

```
int number = 0;  
  
menu_Printf(menu, "How many? : @fd2[#####]\n",  
    &number, &pint_funcs,  
    "Enter the number of them (0 - 15)", "(0,15)");
```

### Description

Allows for the calculator style input of a long integer. It adds digits at the right and pushes others to the left. The  and  keys delete the right most digit and pull the rest to the right. The  clears the field. The variable for **plong\_funcs** is a long *not* an unsigned long.

### Variable Type

long \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct plong_funcs = {  
    num_fenter,  
    long_fexit,  
    pnum_fkey,  
    long_senter,  
    long_sexit,  
    sizeof(long)  
};
```

### Source File

*fnpnum.c*

### See Also

**long\_funcs, clong\_funcs, slong\_funcs, digit\_funcs**

### Example

```
long value = 0L;  
  
menu_Printf(menu, "Value : @fd2[#####]",  
    &value, &plong_funcs,  
    "Enter a value 0 to 1500000", "(0,1500000)");
```

### Description

Allows for the exclusive selection of one field from a group of fields, each of which has the same field name. Requires at least one writeable position (for a check mark).

All fields of the same name are considered a radio group and only one selection per group is allowed.

To have **radio\_funcs** simulate real radio buttons (i.e., one field must be chosen and only one field may be chosen), you must initialize all variables in the group to FALSE except one (the default choice), which you must initialize to TRUE.

Remove the **#define ALWAYS\_ONE** line if you wish to allow the user to choose none of the fields.

### Variable Type

boolean \*

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct radio_funcs = {
    stdNoCur_fenter,
    std_fexit,
    radio_fkey,
    check_senter,
    check_sexit,
    sizeof(boolean)
};
```

### Source File

*fnradio.c*

### See Also

**radiobob\_funcs**



## Example

```
menu_type menu;
boolean   stooge[4];
int       top_stooge;
char      *stooge_name[4] = {"Moe", "Larry", "Curly", "Shemp"};

/* initialize the stooge array */
stooge[0] = TRUE; /* Moe is the default */
for (top_stooge = 1; top_stooge < 4; top_stooge++) {
    stooge[top_stooge] = FALSE;
}

/* ... */

menu = menu_Open();
menu_Printf(menu, "Favorite stooge:\n\n");
menu_Printf(menu, "@f{stooge}[ # Moe  ]\n", stooge+0,
    &radio_funcs);
menu_Printf(menu, "@f{stooge}[ # Larry ]\n", stooge+1,
    &radio_funcs);
menu_Printf(menu, "@f{stooge}[ # Curly ]\n", stooge+2,
    &radio_funcs);
menu_Printf(menu, "@f{stooge}[ # Shemp ]\n", stooge+3,
    &radio_funcs);
menu_Printf(menu, "    @f[ ok ]", NULL, &menu_funcs);
menu_Flush(menu);

/* ... */

/* figure out which stooge was picked */
for (top_stooge = 0; top_stooge < 4; top_stooge++) {
    if (stooge[top_stooge] == TRUE) {
        printf("\n%s is top stooge!\n", stooge_name[top_stooge]);
    }
}
```

### Description

Allows for the exclusive selection of one field from a group of fields, all of which are the exclusive contents of a bob. Requires at least one writeable position (for a check mark).

All fields in the bob are considered a radio group and only one selection per group is allowed.

The field that owns the bob should use **bobint\_funcs**, which uses an int variable. Initialize this variable to the default selection's field number plus one. If there is no default, initialize the variable to zero.

Remove the **#define ALWAYS\_ONE** line if you wish to allow the user to choose none of the fields.

### Variable Type

VOID

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct radiobob_funcs = {
    stdNoCur_fenter,
    std_fexit,
    radiobob_fkey,
    radiobob_senter,
    check_sexit,
    0
};
```

### Source File

*fnradiob.c*



### See Also

**radio\_funcs**, **bobint\_funcs**

## Example

```
menu_type    menu, radiomenu;
sed_type     sed, radiosed;
int          top_stooge = 0; /* must initialize this */
char         *stooge_name[4] = {"Moe", "Larry", "Curly", "Shemp"};
/* ... */
/* create the radio button bob */
radiomenu = menu_Open();
menu_Printf(radiomenu, "@f[ # Moe  ]\n", NULL,
    &radiobob_funcs);
menu_Printf(radiomenu, "@f[ # Larry ]\n", NULL,
    &radiobob_funcs);
menu_Printf(radiomenu, "@f[ # Curly ]\n", NULL,
    &radiobob_funcs);
menu_Printf(radiomenu, "@f[ # Shemp ]", NULL,
    &radiobob_funcs);
menu_Flush(radiomenu);
radiosed = sed_Open(radiomenu);
sed_SetSpecial(radiosed, spc_Embed);
/* ... */
/* create the parent sed */
menu = menu_Open();
menu_Printf(menu, "Favorite stooge:\n\n");
menu_Printf(menu, "@fbd[]", &top_stooge, &bobint_funcs,
    sed_CreateBob(radiosed, BOB_DEPENDENT), "Choose a stooge");
/* ... after disp_Close */
if (top_stooge != 0) {
    printf("\n%s is top stooge!\n", stooge_name[top_stooge - 1]);
}
else {
    printf("\nNo choice was made\n");
}
```

### Description

Displays and enters an integer value from 1 to 10 as a horizontal bar of asterisks. The field must have at least 10 writeable positions. Pressing any numeric key assigns that value to the field (the 0 key represents 10). The  key decrements the field's value and the  key increments it.

### Variable Type

int \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string

### Field Function Structure

```
field_funcs_struct range_funcs = {
    stdNoCur_fenter,
    std_fexit,
    range_fkey,
    range_senter,
    FNULL,
    sizeof(int)
};
```

### Source File

*fnrange.c*

### See Also




**int\_funcs, sint\_funcs, digit\_funcs**

### Example

```
int muffins = 10;

menu_Printf(menu, "Ted's muffins : @fd[#####]",
&muffins, &range_funcs, "How many muffins has Ted eaten?");
```

**Description**

Allows the string-like entry of a double floating point value. The  and  keys move through the field and the  key toggles insert/overwrite mode. The record length should be greater than 12.

**Variable Type**

double \*

**Field Data Pointers**

- 0) Prompt message
- 1) Validation string
- 2) Format string

**Field Function Structure**

```
field_funcs_struct sdouble_funcs = {
    stdBigCur_fenter,
    sdouble_fexit,
    sfloat_fkey,
    sdouble_senter,
    sdouble_sexit,
    sizeof(double)
};
```

**Source File**

*fnsdouble.c*

**See Also**

**double\_funcs, sfloat\_funcs**

**Example**

```
double    val = 10.0;
int       min_writeables = 12;

menu_Printf(menu, "Value : @fd2[@[%d,#]]", &val, &sdouble_funcs,
    "Enter a value from 10 to 20", "(10.0,20.0)",
    min_writeables);
```

**Description**

Allows the entry of a character string without echoing characters to the display. Instead, a mask character ('\*') appears for each typed character.

**Variable Type**

char \*

**Field Data Pointers**

- 0) Prompt message
- 1) Validation string

**Field Function Structure**

```
field_funcs_struct secure_funcs = {
    std_fenter,
    secure_fexit,
    secure_fkey,
    secure_senter,
    FNULL,
    VAR_STRING
};
```

**Source File**

*fnsecure.c*

**See Also**

**string\_funcs, alpha\_funcs**




**Example**

```
char password[50];

password[0] = '\0';

menu_Printf(menu, "Password : @fd[#####]",
    password, &secure_funcs, "What is the password?");
```

**Description**

Allows the string-like entry of a floating point value. The  and  keys move through the field and the  key toggles insert/overwrite mode. The record length should be greater than 7.

**Variable Type**

float \*

**Field Data Pointers**

- 0) Prompt message
- 1) Validation string
- 2) Format string

**Field Function Structure**

```
field_funcs_struct sfloat_funcs = {
    stdBigCur_fenter,
    sfloat_fexit,
    sfloat_fkey,
    sfloat_senter,
    sfloat_sexit,
    sizeof(float)
};
```

**Source File**

*fnsfloat.c*

**See Also**




**sdouble\_funcs, double\_funcs**

**Example**

```
float value = 10.0;

menu_Printf(menu, "Value : @fd2[#####]",
    &value, &sfloat_funcs,
    "Enter a value from 10 to 20", "(10.0,20.0)");
```

### Description

Allows the string-like entry of a integer value. The  and  keys move through the field and the  key toggles insert/overwrite mode.

### Variable Type

int \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct sint_funcs = {  
    stdBigCur_fenter,  
    int_fexit,  
    sint_fkey,  
    sint_senter,  
    sint_sexit,  
    sizeof(int)  
};
```

### Source File

*fnsint.c*

### See Also




**int\_funcs, pint\_funcs, digit\_funcs**

### Example

```
int num = 0;  
  
menu_Printf(menu, "How many? : @fd2[#####]",  
    &num, &sint_funcs,  
    "Enter the number of them (0 - 15)", "(0,15)");
```



### Description

Allows the string-like entry of a long integer value. The  and  keys move through the field and the  key toggles insert/overwrite mode.

### Variable Type

long \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct slong_funcs = {
    stdBigCur_fenter,
    long_fexit,
    sint_fkey,
    slong_senter,
    slong_sexit,
    sizeof(long)
};
```

### Source File

*fnslong.c*

### See Also



**long\_funcs, plong\_funcs, clong\_funcs, money\_funcs, cmoney\_funcs**

### Example

```
long val = 0L;

menu_Printf(menu, "Value : @fd2[#####]", &val, &slong_funcs,
    "Enter a value 0 to 1500000", "(0,1500000)");
```

### Description

Controls all the behavior that slug menus exhibit. It checks each field for a bob and passes control when the  key is pressed. If the field has no bob, then the integer associated with that field (assumed to be contained in a string attached to the field's second generic data pointer) is returned through the original **slug\_Go** call. Pressing  returns up one level.

### Variable Type

VOID

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct slug_funcs = {  
    stdNoCur_fenter,  
    std_fexit,  
    slug_fkey,  
    FNULL,  
    FNULL,  
    0  
};
```




### Source File

*fnslug.c*

### See Also

**framer\_funcs**

### Description

Allows the entry of an alphanumeric character string containing standard ASCII characters (32 - 126). The  and  keys move through the field and the  key toggles insert/overwrite mode. The cursor is enlarged in insert mode.

### Variable Type

char \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct string_funcs = {  
    stdBigCur_fenter,  
    string_fexit,  
    string_fkey,  
    string_senter,  
    string_sexit,  
    VAR_STRING  
};
```

### Source File

*fnstring.c*

### See Also

**xstring\_funcs, alpha\_funcs, char\_funcs**

### Example

```
char answer[50];  
  
answer[0] = '\0';  
  
menu_Printf(menu, "To be or not to be: that is the question.\n");  
menu_Printf(menu, "Your answer: @fd2[#####]",  
    answer, &string_funcs,  
    "(to be, not to be)", "to be,not to be");
```

**Description**

Allows full text editing on a string of text.

**ted\_funcs** supports the following keys:

<b>Esc</b>	Quit editing.
<b>Gray -</b>	Cut the marked region into a buffer.
<b>Gray +</b>	Copy the marked region into a buffer.
<b>Del</b>	Delete the marked area if there is one, else delete the character under the cursor.
<b>Ins</b>	Paste the cut buffer into the text.
<b>Alt-M</b>	Mark region by rows.
<b>Alt-C</b>	Mark region by columns.
<b>Alt-S</b>	Search for a string.
<b>Alt-R</b>	Search for a string and replace it.
<b>↑</b>	Moves the cursor up one character.
<b>↓</b>	Moves the cursor down one character.
<b>→</b>	Moves the cursor right one character.
<b>←</b>	Moves the cursor left one character.
<b>Home</b>	Moves to beginning of the current line.
<b>End</b>	Moves to the end of the current line.
<b>PgUp</b>	Moves up one page.
<b>PgDn</b>	Moves down page.
<b>Ctrl-PgUp</b>	Goes to the beginning of the text.
<b>Ctrl-PgDn</b>	Goes to the end of the text.
<b>←</b>	Delete the character to the left of cursor.
<b>Tab</b>	Inserts a tab character.

## Variable Type

char \*

## Field Data Pointers

- 0) Prompt message
- 1) Used by C-scape for the ted's cut buffer; do not allocate a variable for this purpose.

## Field Function Structure

```
field_funcs_struct ted_funcs = {  
    ted_fenter,  
    std_fexit,  
    ted_fkey,  
    ted_senter,  
    ted_sexit,  
    VAR_TED  
};
```

## Source File

*fn ted.c*

## Example

```
sed_type ted;  
char note[100];  
  
note[0] = '\0';  
  
menu = menu_Open();  
menu_Printf(menu, "@fd2[]", note, &ted_funcs, NULL, NULL);  
menu_Flush(menu);  
ted = sed_Open(menu);  
sed_SetBorder(ted, bd_cua);  
ted_SetMaxSize(ted, 100L);  
  
sed_Go(ted);
```

### Description

Allows the entry of a time in the form HH:MM:SS. The field should have only six writeable positions. The results are stored in the time portion of a **tm struct**. Refer to the chapter “Higher Level Functions” in the *C-scape Manual* for information about **tm structs**.

### Variable Type

```
struct tm *
```

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct time_funcs = {  
    std_fenter,  
    time_fexit,  
    time_fkey,  
    time_senter,  
    time_sexit,  
    sizeof(struct tm)  
};
```

### Source File

*fntime.c*

### See Also

**date\_funcs, datet\_funcs, timet\_funcs**

### Example

```
#include "cstime.h"          /* Required by the time functions */  
  
    struct tm bedtime;  
  
    tm_Now(&bedtime);        /* Set time from system clock */  
  
    menu_Printf(menu, "What is your bedtime? @f[##:##:##]",  
        &bedtime, &time_funcs);
```

### Description

Allows the entry of a time in the form HH:MM:SS. The field should have only six writeable positions. The results are stored in the time portion of a `TIME_T` variable. Refer to the chapter “Higher Level Functions” in the *C-scape Manual* for information about C-scape time functions.

### Variable Type

`TIME_T`

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct timet_funcs = {
    std_fenter,
    timet_fexit,
    time_fkey,
    timet_senter,
    timet_sexit,
    sizeof(TIME_T)
};
```

### Source File

*fn`timet`.c*

### See Also

**date\_funcs, datet\_funcs, time\_funcs**

### Example

```
#include "cstime.h"          /* Required by the time functions */

TIME_T bedtime;

ott_Now(&bedtime);          /* Set time from system clock */

menu_Printf(menu, "What is your bedtime? @f[##:##:##]",
    &bedtime, &timet_funcs);
```

### Description

Allows the user to step through a list of choices by pressing the `Space Bar`. The field function's choices are separated by commas in a string attached to the field's second generic data pointer. **toggle\_funcs** copies the selected choice into the field's variable. The field's variable should be large enough to contain the string and its terminating `'\0'` character.

### Variable Type

char \*

### Field Data Pointers

- 0) Prompt message
- 1) Choice list string

### Field Function Structure

```
field_funcs_struct toggle_funcs = {
    stdNoCur_fenter,
    std_fexit,
    toggle_fkey,
    string_senter,
    string_sexit,
    VAR_STRING
};
```

### Source File

*fmtoggle.c*

### See Also

**list\_funcs, togint\_funcs**

### Example

```
char beatle[10];

beatle[0] = '\0';

menu_Printf(menu, "Choose your Beatle : @fd2[#####]", beatle,
    &toggle_funcs, "Space bar displays choices",
    "John,Paul,George,Ringo");
```



### Description

Allows the user to cycle through a list of choices by pressing Space Bar. The field's second data pointer contains choice definitions, with commas separating them. Copies the final choice into the field's variable. The commas do not appear to the user.

### Variable Type

int \*

### Field Data Pointers

- 0) Prompt message
- 1) Choice list string

### Field Function Structure

```
field_funcs_struct togint_funcs = {
    stdNoCur_fenter,
    std_fexit,
    togint_fkey,
    togint_senter,
    togint_sexit,
    sizeof(int)
};
```

### Source File

*fmtogint.c*

### See Also

**list\_funcs, toggle\_funcs**




### Example

```
char marxbro[10];

marxbro[0] = '\0';

menu_Printf(menu, "Choose your Marx brother : @fd2[#####]",
    marxbro, &toggle_funcs, "Space bar displays choices",
    "Groucho,Chico,Harpo,Zeppo");
```

### Description

Allows the entry of a character string that may contain extended ASCII characters (127 - 255). The  and  keys move through the field and the  key toggles between insert and overwrite mode. The cursor is enlarged in insert mode.

### Variable Type

char \*

### Field Data Pointers

- 0) Prompt message
- 1) Validation string
- 2) Format string

### Field Function Structure

```
field_funcs_struct xstring_funcs = {  
    stdBigCur_fenter,  
    string_fexit,  
    xstring_fkey,  
    string_senter,  
    string_sexit,  
    VAR_STRING  
};
```

### Source File

*fnstring.c*

### See Also

**string\_funcs, alpha\_funcs**

### Example

```
char text[50];  
text[0] = '\0';  
  
menu_Printf(menu, "Enter extended ASCII characters by holding\n");  
menu_Printf(menu, "down the ALT key and typing a three digit\n");  
menu_Printf(menu, "value on the numeric keypad.\n\n");  
menu_Printf(menu, "Try it : @f[#####]", text, &xstring_funcs);
```

### Description

Allows the entry of a boolean value, but displays “Yes” for TRUE and “No” for FALSE. The `[Space Bar]` toggles the field, and the `[Y]`, `[y]`, `[N]`, `[n]` keys set it appropriately. The field should contain enough writeable positions to display the “Yes” and “No” strings.

You can use different strings for “Yes” and “No” by setting the `yes_ptr` and `no_ptr` elements of the `ocountry_struct`:

```
ocountry.yes_ptr = "Ja";
ocountry.no_ptr = "Nein";
```

Refer to the *C-scape Manual* for more on the `ocountry_struct`.

### Variable Type

boolean \*

### Field Data Pointers

- 0) Prompt message

### Field Function Structure

```
field_funcs_struct yesno_funcs = {
    stdNoCur_fenter,
    yesno_fexit,
    yesno_fkey,
    yesno_senter,
    yesno_sexit,
    sizeof(boolean)
};
```

### Source File

*fnyesno.c*

### See Also

`toggle_funcs`

### Example

```
boolean truth;

menu_Printf(menu, "Is it true? @fd[###]", &truth, &yesno_funcs,
    "The Space bar toggles.");
```



## Appendix C: C-scape Error Messages

---

<i>Error</i>	<i>Description</i>
1	<b>field_Close</b> : passed bad field.
2	<b>field_FirstChar</b> : passed bad field.
3	<b>field_GetMerge</b> : passed bad field.
4	<b>field_GetRecord</b> : passed bad field.
6	<b>field_LastChar</b> : passed bad field.
7	<b>field_NextChar</b> : passed bad field.
8	<b>field_NextChar</b> : passed bad record position.
9	<b>field_PrevChar</b> : passed bad record position.
10	<b>field_SetChar</b> : bad field position or field contains no writeable positions.
11	<b>field_SetChar</b> : passed bad field.
12	<b>field_SetString</b> : passed bad field.
13	<b>field_GetData</b> : passed bad field.
14	<b>field_SetData</b> : passed bad field.
20	<b>draw_field</b> : illegal call. This is an internal error.
22	<b>sd_goto_char</b> : passed negative field position. ( <b>sd_goto_char</b> is an internal C-scape routine used to move the cursor within a field.)
23	<b>sd_goto_char</b> : passed bad sed. ( <b>sd_goto_char</b> is an internal C-scape routine used to move the cursor within a field.)
24	<b>sd_overwrite</b> : passed bad sed. ( <b>overwrite</b> is an internal C-scape routine used to write characters to a field.)
25	<b>sed_Pop</b> : passed bad sed.
27	<b>sd_refresh_field</b> : passed out-of-range field number.
28	<b>sd_refresh_field</b> : no fields in sed.
29	<b>sed_Draw</b> : passed bad sed. ( <b>sed_Draw</b> is an internal routine used by <b>sed_Repaint</b> , <b>sed_RepaintFields</b> , <b>sed_Update</b> , and <b>sed_UpdateFields</b> .)
30	<b>sd_scroll</b> : passed bad sed.
40	<b>menu_Destroy</b> : passed bad menu.
41	<b>menu_Flush</b> : passed bad menu.
42	<b>menu_GetDownField</b> : passed out-of-range field number.
43	<b>menu_GetDownField</b> : passed bad menu.

<i>Error</i>	<i>Description</i>
44	<b>menu_GetLeftField:</b> passed out-of-range field number.
45	<b>menu_GetLeftField:</b> passed bad menu.
46	<b>menu_GetRightField:</b> passed out-of-range field number.
47	<b>menu_GetRightField:</b> passed bad menu.
51	<b>menu_GetUpField:</b> passed out-of-range field number.
52	<b>menu_GetUpField:</b> passed bad menu.
53	<b>menu_Printf:</b> percent buffer overflow. A percent (%) expansion within a format string larger than 511 characters has occurred. These must be less than 511 characters.
54	<b>menu_Ok:</b> menu has been closed. An attempt to use a menu object that has already been closed has occurred. Either do not close the menu, or open a new menu.
55	<b>menu_Printf:</b> @c arguments too long. The number of characters within the brackets of an @c command has exceeded 127.
56	<b>menu_Printf (percent) (menu.c):</b> unanticipated conversion character. The percent (%) character has been used in a format string without a recognizable conversion character (such as "%z").
58	<b>menu_Printf:</b> unexpected end of format string. The end of the format string has been reached by the parser before it was expected. Usually this occurs when the closing ']' of a command is missing. This can also occur when an '@' is found with no command or quoted character following it. Quoting the '@' with another '@' will usually solve the problem.
59	<b>menu_Printf (repeat):</b> bad repeat arguments. The repeat syntax has been given bad arguments (either no comma was found or the repeat string was of zero length).
60	<b>menu_Printf:</b> field too long. A field longer than 1000 (MAX_FIELD_LEN) characters was encountered.
61	<b>menu_Printf:</b> passed bad menu.
62	<b>menu_Printf:</b> @p arguments too long. The number of characters within the brackets of an @p command has exceeded 127.
63	<b>menu_Printf:</b> repeat arguments too long. The number of characters within the brackets of a repeat '@[' command has exceeded 127.

<i>Error</i>	<i>Description</i>
64	<b>menu_Printf</b> : out-of-range repeat count. The repeat count is either negative or greater than 127.
65	<b>menu_Printf</b> : unanticipated state. This is an internal error in the <b>menu_Printf</b> parser.
69	<b>set_field</b> ( <i>menu.c</i> ): passed bad menu.
70	<b>menu_Printf</b> : missing '[' in @a command.
71	<b>menu_Printf</b> : @a arguments too long. The number of characters within the brackets of an @a command has exceeded 127.
74	<b>menu_Printf</b> : bad '@' command. An invalid character was found following '@' in the format string.
75	<b>menu_Printf</b> : missing '[' in @p command.
76	<b>menu_Printf</b> : missing '[' in @c command.
77	<b>menu_Printf</b> : missing '[' in @f command.
79	<b>menu_Printf</b> : field name too long.
80	<b>menu_MoveField</b> : passed bad menu.
81	<b>menu_MoveField</b> : passed bad field number.
82	<b>menu_DeleteField</b> : passed bad menu.
83	<b>menu_DeleteField</b> : passed bad field number.
84	<b>menu_DeleteRows</b> : passed bad menu.
85	<b>menu_DeleteRows</b> : passed bad arguments.
86	<b>menu_InsertRows</b> : passed bad menu.
87	<b>menu_InsertRows</b> : passed bad arguments.
88	<b>menu_SetTB</b> : passed bad menu.
89	<b>menu_SetTB</b> : passed bad text.
90	<b>menu_SwapFields</b> : passed bad menu.
91	<b>menu_SwapFields</b> : passed bad arguments.
100	<b>sed_Clear</b> : passed bad sed.
101	<b>sed_Close</b> : passed bad sed.
102	<b>sed_DecChar</b> : passed bad sed.
103	<b>sed_DecField</b> : passed bad sed.
104	<b>sed_GetColors</b> : passed bad pointers. One or more of the pointers passed is equal to NULL.



<i>Error</i>	<i>Description</i>
105	<b>sed_GetColors</b> : passed bad sed.
106	<b>sed_GotoFirstField</b> : passed bad sed.
107	<b>sed_GotoLastField</b> : passed bad sed.
108	<b>sed_Go</b> : passed bad sed.
109	<b>sed_GetPosition</b> : passed bad sed.
110	<b>sed_GotoChar</b> : passed bad field position.
111	<b>sed_GotoChar</b> : passed too large field position.
112	<b>sed_GotoChar</b> : passed bad sed.
113	<b>sed_GotoField</b> : passed bad field number.
114	<b>sed_GotoField</b> : passed bad sed.
115	<b>sed_Goto</b> : no fields in sed.
116	<b>sed_Goto</b> : passed negative field number.
117	<b>sed_Goto</b> : passed bad sed.
118	<b>sed_IncChar</b> : passed bad sed.
119	<b>sed_IncField</b> : passed bad sed.
120	<b>sed_MarkField</b> : passed bad sed.
121	<b>sed_MoveGField</b> : passed bad direction. This is an internal error.
122	<b>sed_MoveGField</b> : passed bad sed. <b>sed_MoveGField</b> is the function that handles grid movement between fields (UpField, DownField, etc.).
123	<b>sed_Overwrite</b> : passed bad sed.
124	<b>sed_OK</b> : passed a closed sed. An attempt has been made to use a sed that been closed.
125	<b>sed_PageRight</b> : passed bad sed.
126	<b>sed_PageDown</b> : passed bad sed.
127	<b>sed_PageLeft</b> : passed bad sed.
128	<b>sed_PageUp</b> : passed bad sed.
129	<b>sed_PushLeft</b> : passed bad sed.
130	<b>sed_PushRight</b> : passed bad sed.
131	<b>sed_PullLeft</b> : passed bad sed.
132	<b>sed_PullRight</b> : passed bad sed.
133	<b>sed_RepaintField</b> : passed bad sed.
134	<b>sed_SetBorder</b> : passed bad sed.

<i>Error</i>	<i>Description</i>
136	<b>scroll_adjust</b> : passed bad sed. <b>scroll_adjust</b> is an internal C-scape routine called during movement between fields.
137	<b>sed_ScrollRight</b> : passed bad sed.
138	<b>sed_ScrollDown</b> : passed bad sed.
139	<b>sed_ScrollLeft</b> : passed bad sed.
140	<b>sed_ScrollUp</b> : passed bad sed.
141	<b>sed_SetColors</b> : passed bad sed.
142	<b>sed_GetFieldColors</b> : passed bad pointers. One or more of the pointers passed are equal to NULL.
143	<b>sed_GetFieldColors</b> : passed bad sed.
144	<b>sed_SetHeight</b> : passed bad sed.
145	<b>sed_SetHeight</b> : passed bad height.
146	<b>sed_SetPosition</b> : passed bad sed.
147	<b>sed_SetRecord</b> : passed bad sed.
148	<b>sed_SetPosition</b> : passed negative position.
149	<b>sed_SetWidth</b> : passed bad sed.
150	<b>sed_SetWidth</b> : passed bad width.
170	<b>tb_Close</b> : bad text buffer.
172	<b>tb_DrawLines</b> : passed bad text buffer.
175	<b>tb_Puts</b> : passed out-of-range coordinates. <b>tb_Puts</b> is called by <b>menu_Printf</b> whenever it adds text to the text buffer. This error indicates that an attempt was made to place text at negative coordinates.
176	<b>tb_Puts</b> : passed bad text buffer.
178	<b>tb_SetData</b> : passed bad text buffer.
179	ted: no cursor move method set. A ted cursor movement routine was used without first calling <b>ted_StartWorking</b> .
180	<b>varsize</b> : illegal use of field function. A field function whose <i>varsize</i> element is VAR_INVALID has been used with a sled or screen file.
190	<b>sfile_LoadSed</b> : passed bad screen file. The screen file has not been opened correctly. Check the return value of <b>sfile_Open</b> for NULL.
191	<b>sfile_Close</b> : passed bad screen file.

<i>Error</i>	<i>Description</i>
1001	<b>disp_GetFont:</b> passed bad display manager. The device interface has not been initialized.
1002	<b>wmgr_Init:</b> passed bad display manager. The device interface has not been initialized.
1003	<b>wmgr_Wrapup:</b> passed bad display manager. The device interface has not been initialized.
1004	<b>wmgr_CreateWin:</b> passed bad display manager. The device interface has not been initialized.
1005	<b>wmgr_ExposePixBox:</b> passed bad display manager. The device interface has not been initialized.
1006	<b>wmgr_ExposePixBox:</b> bad tiler data pointer. The window manager could not allocate memory.
1007	<b>kb_Record:</b> passed bad display manager. The device interface has not been initialized.
1008	<b>kb_Idle:</b> passed bad display manager. The device interface has not been initialized.
1009	<b>disp_GetHeight:</b> passed bad display manager. The device interface has not been initialized.
1010	<b>disp_GetWidth:</b> passed bad display manager. The device interface has not been initialized.
1011	<b>disp_Clear:</b> passed bad display manager. The device interface has not been initialized.
1012	<b>disp_Repaint:</b> passed bad display manager. The device interface has not been initialized.
1013	<b>disp_GetCurrent:</b> passed bad display manager. The device interface has not been initialized.
1014	<b>disp_SetCurrent:</b> passed bad display buffer.
1015	<b>win_MakeCurrent:</b> found bad current window.
1016	<b>win_MakeCurrent:</b> passed bad window.
1017	<b>win_MakeCurrent:</b> called within a mouse handler.
1018	<b>win_Close:</b> passed bad window.
1019	<b>win_Class:</b> WINM_CLOSE: window has bad parent.
1020	<b>obj_Open:</b> OBJM_GETDATSIZE: uninitialized od. An object class has not set its od_size.

<i>Error</i>	<i>Description</i>
<b>1021</b>	<b>obj_Open:</b> OBJM_GETDATSIZE: uninitialized xd. An object class has not set its xd_size.
<b>1022</b>	<b>obj_Open:</b> OBJM_GETDATSIZE: uninitialized id. An object class has not set its id.
<b>1100</b>	<b>OGL:</b> call to uninitialized OGL routine An <b>ogl_</b> routine has been called without calling <b>ogl_Init</b> .

Codes not listed in this appendix appear in the source files *cerror.h* and *oakerror.h*.

# Reader's Comment Form

## C-scape Function Reference

M-2202.4 September 1992

Liant Software Corporation welcomes your comments on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

**Please rate this manual in the following areas:**

	Excellent	Good	Fair	Poor
Accuracy (product works as described)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Clarity of instructions	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Completeness of information	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ease of finding information	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Usefulness of examples	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Usefulness of Figures and Tables	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Logical organization	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Overall appearance of page design	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Quality of binding	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Quality of print	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Quantity and quality of screens	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**What features did you like most about this manual?** \_\_\_\_\_

---

---

---

**Do you need more information? If so, on what topic?** \_\_\_\_\_

---

---

---

**Additional comments or suggestions for improvement:** \_\_\_\_\_

---

---

---

**Did you find any errors in this manual?**

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____

**Please send or fax your comments to:**

Liant Software Corporation  
Technical Documentation Department  
959 Concord Street  
Framingham, MA 01701-4613

Tel: (508) 872-8700

Fax: (508) 626-2221

**If you would like a reply, please provide the following information:**

Name: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

City: \_\_\_\_\_ State/Province: \_\_\_\_\_

Country: \_\_\_\_\_ Zip/Postal Code: \_\_\_\_\_

Phone Number: (\_\_\_\_\_) \_\_\_\_\_

Fax Number: (\_\_\_\_\_) \_\_\_\_\_

**Thank you for helping us improve our documentation.**

---

# LIANT

Liant Software Corporation  
959 Concord Street  
Framingham, MA 01701  
U.S.A.  
(508) 872-8700



---

**Version 4**

**FUNCTION REFERENCE**

---

**LIANT**