

**DeskMate Development System
Technical Reference
03.05 Replacement Pages**

*These pages
have been
replaced*

(2) Clipboard Manager

Replace pages 2-3 thru 2-8.

(3) Communication Manager

Replace pages 3-3 thru 3-8.

(4) Component Manager

Replace pages 4-1 thru 4-4.

Add new pages 4-1.1, 4-3.1

Replace pages 4-11 thru 4-14.

Add new pages 4-11.1, 4-14.1 thru 4-14.3

Replace pages 4-25 thru 4-36

Replace pages 4-41 thru 4-42

(5) Configuration Manager

Completely replace this section.

(6) Database Resource

Replace pages 6-1 thru 6-2.

Replace pages 6-11 thru 6-12.

Replace pages 6-45 thru 6-46.

(7) Desk Executive

Replace pages 7-23 thru 7-36.

Add page 7-23.1

Replace pages 7-47 thru 7-50.

(8) Dialog Box Manager

Replace pages 8-1 thru 8-2.

Replace pages 8-7 thru 8-8.

(9) Environment Manager

Replace page 9-7 thru 9-8.

(10) Event Manager

Replace page 10-11 thru 10-14.

*Note: Check the header files for bugs;
there are some. Always verify the header
with the documentation before using the
function.*

*Note: All pointers passed to deskmate must
address static RAM, no auto strings. Heap
should be OK, but must stay allocated after
passing.*

**DeskMate Development System
Technical Reference
03.05 Replacement Pages**

(11) File I/O Manager

Replace pages 11-19 thru 11-20.
Replace pages 11-35 thru 11-36.
Replace pages 11-49 thru 11-50.
Replace pages 11-59 thru 11-60.

(12) Form Resource

Replace the table of contents. The FAR FORM table of contents is after page 12-54.
Replace page 12-1 thru 12-4.
Replace page 12-15 thru 12-16.
Replace page 12-43 thru 12-44.

Completely replace the FAR FORM Resource section including the table of contents. The pages are numbered 12-55 thru 12-99.

(13) Intelligent Help Manager

Replace pages 13-3 thru 13-4.

(16) Library Functions

Replace pages 16-1 thru 16-2.
Add pages 16-1.1 thru 16-1.2.
Replace pages 16-15 thru 16-16.

(17) Menu Bar Manager

Replace pages 17-1 thru 17-2.

(20) Print Managers

Replace pages 20-1 thru 20-2.
Replace pages 20-7 thru 20-12.
Replace pages 20-17 thru 20-18.
Replace pages 20-21 thru 20-22.

(25) Video Manager

Add new pages 25-8.1 thru 25-8.1.
Replace pages 25-9 thru 25-12.
Replace pages 25-77 thru 25-78.
Replace pages 25-95 thru 25-98.

(26) Window Manager

Replace pages 26-1 thru 26-2.
Add new page

**DeskMate Development System
Technical Reference
03.05 Replacement Pages**

(27) Font Resource

Add New Section.

(28) Screen Save Resource

Add New Section.

DeskMate Development System Technical Reference 03.05 Updates

This document lists the pages in the DeskMate Technical Reference which were not reprinted for the DeskMate Development System 03.05 Update because of the number of pages required and the significance of the change.

The majority of the changes listed below are references to "DeskMate 03.03.0x ONLY" changing to "DeskMate 03.03 and later".

Database Resource

Database Resource

Page: 6-13 thru 6-59

Function Name: General Description/Notes, All database manager function calls.

Correction: All references to "1989 DeskMate 03.03.0x" should be changed to "DeskMate 03.03 and later".

Database Resource

Page: 6-10

Function Name: handle_buf structure

Correction: The utilized by should contain SAVE_PAGE_SETUP and GET_PAGE_SETUP.

Database Resource

Page: 6-24

Function Name: db_bind_read

Correction: Add the following paragraph to the Special Notes section "**db_bind_read** is only available when running on DeskMate 03.03 or later."

Database Resource

Page: 6-25

Function Name: db_end_read

Correction: Add the following paragraph to the Special Notes section "**db_end_read** is only available when running on DeskMate 03.03 or later."

Desk Executive

Desk Executive

Page: 7-9

Function Name: db_bind_read

Correction: Add the following paragraph to the Special Notes section "**db_bind_read** is only available when running on DeskMate 03.03 or later."

Desk Executive

Page: 7-10

Function Name: db_bind_read

Correction: All references to "1989 DeskMate 03.03.0x" were changed to "DeskMate 03.03 and later".

Desk Executive

Page: 7-11

Function Name: db_end_read

**DeskMate Development System
Technical Reference
03.05 Updates**

Correction: Add the following paragraph to the Special Notes section "db_bind_read is only available when running on DeskMate 03.03 or later."

Desk Executive

Page: 7-12

Function Name: END_DB

Correction: Add the following paragraph to the Special Notes section "END_DB requires the application to link with DeskMate 03.03 or later libraries (DM.LIB or DMMED.LIB)."

Desk Executive

Page: 7-17

Function Name: dm_bind_resource

Correction: Added the following paragraph to the Special Notes section. "dm_bind_resource is only available when running DeskMate 03.03 and later."

Desk Executive

Page: 7-18

Function Name: dm_clear_resource

Correction: Added the following paragraph to the Special Notes section. "dm_clear_resource is only available when running DeskMate 03.03 and later."

Desk Executive

Page: 7-19

Function Name: dm_compat

Correction: Added the following paragraph to the Special Notes section. "dm_compat is only available when running DeskMate 03.03 and later."

Desk Executive

Page: 7-37

Function Name: dm_temp_resource

Correction: Added the following paragraph to the Special Notes section. "dm_temp_resource is only available when running DeskMate 03.03 and later."

Desk Executive

Page: 7-39

Function Name: csr_access_end

Correction: Added the following paragraph to the Special Notes section. "csr_access_end is only available when running DeskMate 03.03 and later."

Desk Executive

Page: 7-42

Function Name: csr_form_bind_end

Correction: Added the following paragraph to the Special Notes section. "csr_form_bind_end is only available when running DeskMate 03.03 and later."

Desk Executive

Page: 7-43

Function Name: csr_form_bind_init

Correction: Added the following paragraph to the Special Notes section. "csr_form_bind_init is only available when running DeskMate 03.03 and later."

DeskMate Development System Technical Reference 03.05 Updates

Desk Executive

Page: 7-46

Function Name: dmcsr_bind_end

Correction: Added the following paragraph to the Special Notes section. "dmcsr_bind_end is only available when running DeskMate 03.03 and later."

Desk Executive

Page: 7-52 and 7-53

Function Name: prguf_bind_end, prguf_bind_init

Correction: prguf_bind_init requires the application to link with DeskMate 03.03 or later libraries (DM.LIB or DMMED.LIB)."

Environment Manager

Environment Manager

Page: 9-6

Function Name: env_get

Correction: in the Input Parameters section pData should be changed to pDataInfo.

File I/O Manager

File I/O Manager

Page: 11-11

Function Name: dlgbox_Run_this

Correction: Added the following paragraph to the Special Notes section. "dlgbox_Run_this is only available when running DeskMate 03.03 and later."

File I/O Manager

Page: 11-13

Function Name: dosENDFIND

Correction: Added the following paragraph to the Special Notes section. "dosENDFIND is only available when running DeskMate 03.03 and later."

File I/O Manager

Page: 11-14

Function Name: dosFINDFIRST

Correction: Added the following paragraph to the Special Notes section. "dosFINDFIRST is only available when running DeskMate 03.03 and later."

File I/O Manager

Page: 11-15

Function Name: dosFINDNEXT

Correction: Added the following paragraph to the Special Notes section. "dosFINDNEXT is only available when running DeskMate 03.03 and later."

**DeskMate Development System
Technical Reference
03.05 Updates**

File I/O Manager

Page: 11-16

Function Name: dosSTARTFIND

Correction: Added the following paragraph to the Special Notes section. "dosSTARTFIND is only available when running DeskMate 03.03 and later."

File I/O Manager

Page: 11-46

Function Name: fil_read_far

Correction: Added the following paragraph to the Special Notes section. "fil_read_far is only available when running DeskMate 03.03 and later."

File I/O Manager

Page: 11-51

Function Name: fil_write_far

Correction: Added the following paragraph to the Special Notes section. "fil_write_far is only available when running DeskMate 03.03 and later."

File I/O Manager

Page: 11-62 and 11-63

Function Name: prguf_bind_end, prguf_bind_init

Correction: prguf_bind_end and prguf_bind_init requires the application to link with DeskMate 03.03 or later libraries (DM.LIB or DMMED.LIB)."

File I/O Manager

Page: 11-64

Function Name: qfn

Correction: Added the following paragraph to the Special Notes section. "qfn requires the application to link with DeskMate 03.03 or later libraries (DM.LIB or DMMED.LIB)."

File I/O Manager

Page: 11-72

Function Name: valid_filename_wild

Correction: Added the following paragraph to the Special Notes section. "valid_filename_wild is only available when running DeskMate 03.03 and later."

Intelligent Help Manager

Intelligent Help Manager

Page: 13-1

Function Name: General Description/Notes

Correction: Changes the second paragraph to read "All of the calls in this section require the application to link with DeskMate 03.03 or later libraries."

Intelligent Help Manager

Page: 13-3 thru 13-12

Function Name: All of the function calls.

Correction: Add a paragraph to each call which reads "This call requires the application to link with DeskMate 03.03 or later libraries (DM.LIB or DMMED.LIB)."

**DeskMate Development System
Technical Reference
03.05 Updates**

Library Functions

Library Functions

Page: 16-4

Function Name: `about_versions`

Correction: Add a paragraph to in Special Notes section which reads "**about_versions** requires the application to link with DeskMate 03.03 or later libraries (DM.LIB or DMMED.LIB).".

Menu Bar Manager

Menu Bar Manager

Page: 17-12

Function Name: `mb_get_status`

Correction: The following paragraph should be added to the Special Notes section **mb_get_status** requires the application to link with DeskMate 03.03 or later libraries (DM.LIB or DMMED.LIB).

Video Manager

Video Manager

Page: 25-8

Function Name: `csr_load_video_driver`

Correction: Remove the first line which reads "1989 DeskMate 03.03.0x ONLY". Add the following line to the Special Notes section. "**csr_load_video_driver** is only available when running DeskMate 03.03 and later."

Video Manager

Page: 25-43

Function Name: `vid_get_far_screen`

Correction: Remove the first line which reads "1989 DeskMate 03.03.0x ONLY". Add the following line to the Special Notes section. "**vid_get_far_screen** is only available when running DeskMate 03.03 and later."

Video Manager

Page: 25-54

Function Name: `vid_loadable_drivers`

Correction: Remove the first line which reads "1989 DeskMate 03.03.0x ONLY". Add the following line to the Special Notes section. "**vid_loadable_drivers** is only available when running DeskMate 03.03 and later."

Video Manager

Page: 25-55

Function Name: `vid_memory_free`

Correction: Remove the first line which reads "1989 DeskMate 03.03.0x ONLY". Add the following line to the Special Notes section. "**vid_memory_free** is only available when running DeskMate 03.03 and later."

**DeskMate Development System
Technical Reference
03.05 Updates**

Video Manager

Page: 25-56

Function Name: vid_memory_info

Correction: Remove the first line which reads "1989 DeskMate 03.03.0x ONLY". Add the following line to the Special Notes section. "vid_memory_info is only available when running DeskMate 03.03 and later."

Video Manager

Page: 25-57

Function Name: vid_memory_inuse

Correction: Remove the first line which reads "1989 DeskMate 03.03.0x ONLY". Add the following line to the Special Notes section. "vid_memory_inuse is only available when running DeskMate 03.03 and later."

Video Manager

Page: 25-69

Function Name: vid_put_far_screen

Correction: Remove the first line which reads "1989 DeskMate 03.03.0x ONLY". Add the following line to the Special Notes section. "vid_put_far_screen is only available when running DeskMate 03.03 and later."

DeskMate Technical Reference
Version 03.03.00

DeskMate Development System:
Copyright©1988, 1989 Tandy Corporation
All Rights Reserved

DeskMate Development System
03.03.00

About This Kit
DeskMate Style Guide
DeskMate Development Guide
Copyright©1989 Tandy Corporation
All Rights Reserved

DeskMate Technical Reference
Copyright©1988, 1989 Tandy Corporation
All Rights Reserved

Reproduction or use of any portion of these manuals, without express written permission from Tandy Corporation and/or its licensor, is prohibited. While reasonable efforts have been made in the preparation of these manuals to assure their accuracy, Tandy Corporation assumes no liability resulting from any errors in or omissions from these manuals, or from the use of the information contained herein.

DeskMate, *DeskMate Getting Started*, Radio Shack, and Tandy are registered trademarks of Tandy Corporation.

Hayes is a registered trademark of Hayes Microcomputer Products, Inc.

Hercules is a registered trademark of Hercules Computer Technology.

HP LaserJet is a registered trademark of Hewlett-Packard Corporation.

IBM, IBM PC, PS/2, Video Graphics Array, Multi Color Graphics Array, Enhanced Graphics Adapter, and Color Graphics Adapter are registered trademarks of International Business Machines Corporation.

Microsoft, MS-DOS, and CodeView, are registered trademarks of Microsoft Corporation. QuickC is a trademark of Microsoft Corporation.

PC-Link is a registered service mark of Quantum Computer Services, Inc. *PC-Link Connect Guide* is a registered trademark of Quantum Computer Services, Inc.

Periscope is a trademark of The Periscope Company, Inc.

Turbo C, Turbo Assembler, Turbo Debugger and the other Borland tools mentioned herein are trademarks or registered trademark of Borland International, Inc.

10 9 8 7 6 5 4 3 2 1

Table of Contents

Autoload Resource	1
General Description/Notes	1- 1
Structures/Defines	1- 2
autoload_bind_end ...Release the autoload resource.....	1- 4
autoload_bind_init ..Initializes autoload resource.....	1- 5
autoload_setRegister and Delete autoload resource.....	1- 6
 Clipboard Manager	 2
General Description/Notes	2- 1
Defines	2- 2
dm_clear_clipboardClears data from Clipboard.....	2- 3
dm_clipboard_sizeReturns size of Clipboard memory.....	2- 4
dm_free_clip_bufferReturns memory to Clipboard.....	2- 5
dm_get_clip_bufferGets unused space in Clipboard memory.	2- 6
dm_get_clipboard_info ...Gets type, length, & pointer.....	2- 7
dm_set_clipboard_info ...Sets type, length.....	2- 8
 Communication Manager	 3
General Description/Notes	3- 1
Structures/Defines	3- 2
com_callCalls the specified number.....	3- 4
com_closeCloses the current device.....	3- 5
com_discDisconnects the terminal session.....	3- 6
com_get_buffer_info ...Gets com structure information.....	3- 7
com_get_lines_states ..Gets the com port status.....	3- 8
com_get_parmsGets the communication parameters.....	3- 9
com_openOpens the communication device.....	3-10
com_put_charWrites a character to the com device....	3-11
com_put_stringWrites a string to the com device.....	3-12
com_send_breakSends break sequence to com device.....	3-13
com_set_parmsSets the communications parameters.....	3-14
com_test_carrierCheck com port for carrier signal.....	3-15

Component Manager	4
General Description/Notes	4- 1
Structures/Defines	4- 2
Application Defined Components	
General Description/Notes	4-15
Structures/Defines	4-16
cmp_addAdds an application defined component.....	4-17
cmp_closeCloses an application defined component.....	4-19
cmp_deleteDeletes an application defined component.....	4-20
cmp_drawDraws an application defined component.....	4-21
cmp_openOpens an application defined component.....	4-22
cmp_runRuns an application defined component.....	4-23
Pre-Defined Components	
General Description/Notes	4-25
Check box	4-25
Edit field	4-25
Icon button	4-26
List box	4-26
Menu bar	4-26
Push button	4-26
Radio buttons	4-27
Edit field examples	
Single line editfield	4-28
Formatted editfield	4-29
Multi line editfield	4-30
cmp_closeCloses a Pre-Defined component.....	4-31
cmp_drawDraws a Pre-Defined component.....	4-32
cmp_openOpens a Pre-Defined component.....	4-33
cmp_runRuns a Pre-Defined component.....	4-34

Component Utilities

edt_clear	Clears selected edit field text.....	4-36
edt_copy	Copies edit field text to clipboard.....	4-37
edt_cut	Cuts edit field text to clipboard.....	4-38
edt_format_multi	Formats edit field string for wordwrap.....	4-39
edt_form_draw	Adds edit field lines to form.....	4-40
edt_paste	Pastes clipboard to edit field.....	4-41
edt_replace	Replaces edit field text w/specified text....	4-42
lb_sort_list	Sorts the items within a list box.....	4-43

Configuration Manager		5
General Description/Notes		5- 1
Structures/Defines		5- 2
cfg_get_com_data	Gets port and modem settings.....	5- 5
cfg_get_com_parms	Gets communication parameters.....	5- 6
cfg_get_moreaccs	Gets "More" menu option status.....	5- 7
cfg_get_ms_data	Gets mouse configuration data.....	5- 8
cfg_get_prt_data	Gets current printer data.....	5- 9
cfg_get_vid_color	Gets palette information.....	5-10
cfg_get_vid_driver ...	Gets the video driver name.....	5-12
cfg_reset_temp_config	Restores temp configuration settings.....	5-13
cfg_restore_cpi	Replaces printer CPI settings	5-14
cfg_set_com_data	Sets port and modem settings.....	5-15
cfg_set_com_parms	Sets communication parameters.....	5-16
cfg_set_dbl_click	Sets the mouse double click speed.....	5-17
cfg_set_moreaccs	Sets the "More" accessory	5-18
cfg_set_ms_data	Sets mouse configuration data.....	5-19
cfg_set_prt_data	Sets current printer data.....	5-20
cfg_set_temp_config	Saves temp configuration settings.....	5-21
cfg_set_temp_cpi	Temporarily replaces CPI.....	5-22
cfg_set_vid_color	Sets palette information.....	5-23
cfg_write	Writes configuration info.....	5-24

Database Resource	6
General Description/Notes	6- 1
Structures/Defines	6- 3
ADD_ROWAdds a row of data to specified table.....	6-13
ADD_ROW_NO_FLUSH ..Adds a row to specified table w/no write....	6-14
ALTER_TABLEAlters the definition of a table.....	6-15
CLOSE_FILECloses a file.....	6-16
CLOSE_TABLECloses a table.....	6-17
COUNT_RECORDSCounts the number of records.....	6-18
COPY_LAYOUTSCopies selected tables to new file.....	6-19
CREATE_FILECreates a new file.....	6-20
CREATE_TABLECreates a new table within a file.....	6-21
db_bind_endTerminates the bindings to DB manager.....	6-22
db_bind_initLoads the DB manager into memory.....	6-23
db_bind_readBinds to 1989 Database Read Resource.....	6-24
db_end_readTerminate bind to 1989 DB Read Resource.....	6-25
DEFINE_INDEXDefines the sort order.....	6-26
DELETE_MULTI_ROWS .Deletes all rows in a specified table.....	6-27
DELETE_ROWDeletes a specified row.....	6-28
DROP_INDEXDrops an index from a table.....	6-29
DROP_TABLEDrops a table and data from file.....	6-30
END_DB.....Unbinds 1989 DB resources from Manager.....	6-31
END_TEMP_SORTEnds the temporary sort order.....	6-32
FETCH_RECORDFetch a record by record number.....	6-33
FIRST_RECORDFetches a single record at a time.....	6-34
GET_COL_INFOGets column information.....	6-35
GET_COLUMN_NAMES ..Gets the column names in a table.....	6-36
GET_INDEX_INFOGets the index information for a table.....	6-37
GET_MAXGets the highest value for column name.....	6-38
GET_MINGets the lowest value for column name.....	6-39
GET_PAGE_SETUPGets the page setup information.....	6-40
GET_RECORD_LOCATION .Gets previous and next record numbers.....	6-41
GET_TABLE_NAMESGets the table names in a file.....	6-42
INIT_BD.....Binds 1989 DB resources to Manager.....	6-43
LAST_RECORDFetches a single record at a time.....	6-44

MERGE	Merges two tables together.....	6-45
MORE_RECORDS	Gets more records which match query.....	6-46
NEXT_RECORD	Fetches a single record at a time.....	6-48
OPEN_FILE	Opens a file.....	6-49
OPEN_TABLE	Opens a table.....	6-50
PACK_TABLE	Packs a table/Removes deleted records.....	6-52
PREV_RECORD	Fetches a single record at a time.....	6-53
SAVE_PAGE_SETUP	Saves the page setup information.....	6-54
SETUP_QUERY	Sets up query constraints.....	6-55
TEMP_SORT	Defines a temporary sort order.....	6-58
UPDATE_ROW	Updates the row specified.....	6-59
Error Code List		6-60

Desk Executive		7
General Description/Notes		7- 1

Autoload Resource Initialization Calls

autoload_bind_end ...	Release the autoload resource.....	7- 4
autoload_bind_init ..	Initializes autoload resource.....	7- 5

Database Resource Initialization Calls

Database Defines		7- 6
db_bind_end ...	Terminates bindings to database resource....	7- 7
db_bind_init ..	Initializes all database resources.....	7- 8
db_bind_read ..	Binds to 1989 Database Read Resource.....	7- 9
db_end_read ...	Terminates bind to 1989 DB Read Resource....	7-11
END_DB	Unbinds 1989 DB resources.....	7-12
INIT_DB	Binds 1989 DB resources	7-13

Desk Executive Defines.....	7-14
dm_acc_run	Runs an accessory..... 7-15
dm_bind_resource	Determines if resource is in memory... 7-17
dm_clear_resource	Removes resource from memory..... 7-18
dm_compat	Compatibility for 1988 DM 03.02.0x.... 7-19
dm_dir_isithere	Checks existance of file..... 7-21
dm_exec_dont_shed	Prevents code shedding..... 7-22
dm_file_search	Searches entire for file..... 7-23
dm_get_mem	Attempts to allocate DOS memory..... 7-24
dm_inquire_critical_err	Gets the last critical error..... 7-25
dm_inquire_desk_ver	Returns version number of DESK.EXE.... 7-26
dm_inquire_dmconfig	Gets the contents of DMCONFIG..... 7-27
dm_inquire_floppy	Determines if a drive can be read..... 7-28
dm_inquire_product	Gets the product information..... 7-29
dm_inquire_program	Returns current application names..... 7-30
dm_inquire_res	Returns resource names..... 7-31
dm_inquire_res_ver	Returns version number of a resource.. 7-32
dm_inquire_taskid	Returns task information..... 7-33
dm_SetNextApp	Sets name and parameter string..... 7-34
dm_SetOtherTask	Sets name of task for switch..... 7-35
dm_task_count	Tells number of tasks in mem..... 7-36
dm_temp_resource	Loads resource in temporary mode..... 7-37
dm_yield	Allow another task to run..... 7-38
Core Services Resource(CSR) Initialization Calls	
csr_access_end	Terminates current active access..... 7-39
csr_access_init	Initializes CSR for new user access... 7-40
csr_end	Terminates application bindings to CSR 7-41
csr_form_bind_end	Terminates bind to near form resource 7-42
csr_form_bind_init	Bind to near form resource..... 7-43
csr_get_version	Gets the version number of the CSR.... 7-44
csr_init	Initialize application bindings to CSR 7-45
dmcsr_bind_end	Terminates use of CSR routines..... 7-46
dmcsr_bind_init	Used to bind to the CSR..... 7-47
fform_bind_end.....	Terminates far form resource..... 7-48
fform_bind_int.....	Bind to far form resource..... 7-49

guf_bind_endTerminates application bindings to GUF 7-50
 guf_bind_initInitialize application bindings to GUF 7-51
prguf_bind_endTerminates low-level GUF bindings..... 7-52
prguf_bind_initSets up low-level GUF bindings..... 7-53
 Spell Resource Initialization Calls
 spell_bind_end.....Terminates interface to spell checker. 7-54
 spell_bind_init.....Initializes a spell checker session... 7-55
 Thesaurus Resource Initialization Calls
 thes_bind_end.....Terminates interface to thesaurus..... 7-56
 thes_bind_init.....Initializes a thesaurus session..... 7-57

Dialog Box Manager 8
 General Description/Notes 8- 1
 Structures/Defines 8- 2
 dlg_drawDisplays a dialog box..... 8- 7
 dlg_draw_frameDraws a dialog box frame..... 8- 8
 dlg_runRuns a dialog box..... 8- 9

Environment Manager 9
 General Description/Notes 9- 1
 Structures/Defines 9- 2
 env_addAppends specified to existing data..... 9- 3
 env_closeRemoves configuration data from memory..... 9- 4
 env_deleteRemoves all configuration data..... 9- 5
 env_getSetup configuration data for use..... 9- 6
 env_openOpens and loads configuration data..... 9- 7
 env_replaceReplaces configuration data on disk..... 9- 8
 env_writeWrites configuration data to disk..... 9- 9

Event Manager	10
General Description/Notes	10- 1
Structures/Defines	10- 3
event_add_edriverAdds a user defined event driver.....	10- 6
event_add_interpAdds a user defined interpreter.....	10- 7
event_delete_edriver ..Deletes a user defined event driver.....	10- 8
event_delete_interp ..Deletes a user defined interpreter.....	10- 9
event_get_uninterp ...Gets an uninterpreted event.....	10-10
event_purge	Purges event from event queue.....10-11
event_read	Returns an event.....10-12
event_scan	Scans for an event.....10-13
event_write	Writes an event to the event queue.....10-14
 File I/O Manager	 11
General Description/Notes	11- 1
Structures/Defines	11- 3
Current_Disk	Returns the current active disk.....11- 7
Delete_File	Deletes a file.....11- 8
dlgbox_Generic	Displays OPEN/MERGE/LOAD dialog box.....11- 9
dlgbox_Run	Displays the RUN dialog box.....11-10
dlgbox_Run_this	Display RUN DB, prompts filled..... 11-11
dlgbox_SaveAs	Displays the SAVE AS dialog box.....11-12
dosENDFIND.....	Restores old Disk Transfer Area to DTA..11-13
dosFINDFIRST.....	Attempts to locate a file name match....11-14
dosFINDNEXT.....	Locates next file or directory match....11-15
dosSTARTFIND.....	Establishes a new Disk Transfer Area....11-16
file_already_exists ...	Determines if a file exists.....11-17
FillWithDir	Fills a list box with a set of files....11-18
fil_access	Returns file attributes.....11-19
fil_close	Closes a file.....11-20
fil_close_error_msg ...	Displays close error messages.....11-21
fil_commit	Writes buffered file data to disk.....11-22
fil_create	Creates a file.....11-23
fil_create_error_msg ..	Displays create error messages.....11-24
fil_dup	Duplicates a file handle.....11-25

fil_force_close	Forces DOS to close a file.....	11-26
fil_lock_bytes	Locks specified portion of file.....	11-27
fil_lseek	Seeks to an offset in a file.....	11-28
fil_lseek_error_msg ...	Displays lseek error messages.....	11-29
fil_menu_merge	Merges a file.....	11-30
fil_menu_new	Takes the application to a "new" state..	11-32
fil_menu_open	Prompts the user for file to open.....	11-34
fil_menu_page	Runs the page setup accessory.....	11-36
fil_menu_quit	Sets up the datafile so app may exit....	11-37
fil_menu_run	Prompts user to run of alternate app....	11-38
fil_menu_save	Saves applications data in memory.....	11-39
fil_menu_saveas	Prompts user for filename to save data..	11-40
fil_open	Opens a file.....	11-41
fil_open_error_msg ...	Displays open error messages.....	11-43
fil_read	Reads from a file.....	11-44
fil_read_error_msg ...	Displays read error messages.....	11-45
fil_read_far	Reads a file from far memory.....	11-46
fil_unlock_bytes	Unlocks specified portion of a file.....	11-48
fil_write	Writes to a file.....	11-49
fil_write_error_msg ...	Displays write error messages.....	11-50
fil_write_far	Writes a file from far memory.....	11-51
Get_Directory	Gets the current directory.....	11-53
get_drive_map	Gets bitmap of all drives in system.....	11-54
get_free_space	Gets the amount of disk free space.....	11-55
guf_bind_end.....	Terminates bindings to GUF resources....	11-56
guf_bind_init.....	Initializes bindings to GUF resources...	11-57
is_floppy	Determines if a drive is removable.....	11-58
msgbox_SaveChanges	Displays the SAVE CHANGES message box...	11-59
Path_Expand	Expands a partial path.....	11-60
prguf_bind_end.....	Terminate bindings to low-level GUF.....	11-62
prguf_bind_init.....	Initializes bindings to low-level GUF...	11-63
qfn.....	Qualifies a path as existing in drive...	11-64
Rename_File	Renames a file.....	11-66
Select_Disk	Selects the specified drive.....	11-67
Set_CPU_Speed	Sets the CPU speed.....	11-68
Set_Directory	Sets the current directory as specified.	11-69

valid_driveDetermines if specified drive is valid..11-70
 valid_filenameVerifies correctness of filename chars..11-71
valid_filename_wild ...Verifies filename using wildcard char...11-72

Form Manager12
 Near Form Resource General Description/Notes.....12- 1
 Structures/Defines12- 4
 form_add_element ..Adds the specified element to the form.....12-13
 form_add_stroke ...Adds a stroke definition to the form.....12-14
 form_break_object .Reduces one object to constituent objects...12-15
 form_clearClears the form's display and stroke lists..12-16
 form_compressCompress free space from the form.....12-17
 form_copy_element .Copies element to the clipboard.....12-18
 form_copy_group ...Copies group of elements to clipboard.....12-19
 form_cut_element ..Cuts the form element to the clipboard.....12-20
 form_cut_groupCuts a group of elements to clipboard.....12-21
 form_define_group ...Defines a group of elements for the form..12-22
 form_delete_element .Deletes element definition from form.....12-23
 form_delete_group ...Deletes all group elements from form.....12-24
 form_delete_stroke ..Deletes stroke character from form.....12-25
 form_delete_unused_strokes ...Deletes non-required strokes.....12-26
 form_display_region .Displays form elements in a region.....12-27
 form_dup_elementCreates a scaled copy of an element.....12-28
 form_dup_groupCreates a scaled copy of a group.....12-29
 form_element_to_bottomMoves element to bottom of defs....12-30
 form_element_to_topMoves element to top of defs.....12-31
 form_expandExpands the form display list.....12-32
 form_find_element ...Finds an element in a rectangle.....12-33
 form_find_strokeFinds a stroke character.....12-34
 form_get_pointerGets a pointer to element definition.....12-35
 form_get_xy_extents .Gets maximum x,y values for all elements..12-36
 form_group_to_bottomMoves group to bottom of defs.....12-37
 form_group_to_topMoves group to top of defs.....12-38
 form_make_objectMakes an object from elements in group....12-39
 form_mod_attrModifies element attributes.....12-40
 form_mod_grp_attr ...Modifies element attributes in the group..12-41

form_move_element ...Moves and resizes an element.....	12-42
form_move_groupMoves and resizes a group of elements.....	12-43
form_openInitializes the form buffer.....	12-44
form_pasteAdds the clipboard display to the form....	12-45
form_region_empty ...Determines if the form contains elements..	12-46
form_replace_elementReplaces an element definition.....	12-47
form_scroll_downScrolls the form down.....	12-48
form_scroll_leftScrolls the form to the left.....	12-49
form_scroll_right ...Scrolls the form to the right.....	12-50
form_scroll_upScrolls the form up.....	12-51
form_specify_group...Makes a list of element tags into a group.	12-52
form_updateDraws/redraws the current form.....	12-53
Far Form Resource General Description/Notes	12-55
fform_add_element ..Adds a specified element to the form.....	12-56
fform_add_stroke ...Adds a stroke definition to the form.....	12-57
fform_bind_endTerminates forms resource loaded.....	12-58
fform_bind_initBinds to far forms resource.....	12-59
fform_break_object .Reduces one object to constituent objects..	12-60
fform_clearClears the form's display and stroke lists.	12-61
fform_compressCompress free space from the form.....	12-62
fform_copy_element .Copies element to the clipboard.....	12-63
fform_copy_group ...Copies group of elements to clipboard.....	12-64
fform_cut_element ..Cuts the form element to the clipboard.....	12-65
fform_cut_groupCuts a group of elements to clipboard.....	12-66
fform_define_group .Defines a group of elements for the form...	12-67
fform_delete_element .Deletes element definition from form.....	12-68
fform_delete_group ...Deletes all group elements from form.....	12-69
fform_delete_stroke ..Deletes stroke character from form.....	12-70
fform_delete_unused_strokes .Deletes non-required strokes.....	12-71
fform_display_region .Displays form elements in a region.....	12-72
fform_dup_elementCreates a scaled copy of an element.....	12-73
fform_dup_groupCreates a scaled copy of a group.....	12-74
fform_element_to_bottomMoves element to bottom of defs....	12-75
fform_element_to_topMoves element to top of defs.....	12-76
fform_expandExpands the form display list.....	12-77
fform_find_element .Finds an element in a rectangle.....	12-78

<code>fform_find_stroke</code>	..Finds a stroke character.....	12-79
<code>fform_get_extents</code>	..Gets maximum x,y values for all elements...	12-81
<code>fform_get_pointer</code>	..Gets a pointer to element definition.....	12-80
<code>fform_group_to_bottom</code>Moves group to bottom of defs.....	12-82
<code>fform_group_to_top</code>Moves group to top of defs.....	12-83
<code>fform_make_object</code>	..Makes an object from elements in group....	12-84
<code>fform_mod_attr</code>Modifies element attributes.....	12-85
<code>fform_mod_grp_attr</code>	..Modifies element attributes in the group...	12-86
<code>fform_move_element</code>	..Moves and resizes an element.....	12-87
<code>fform_move_group</code>	...Moves and resizes a group of elements.....	12-88
<code>fform_open</code>Initializes the form buffer.....	12-89
<code>fform_paste</code>Adds the clipboard display to the form....	12-90
<code>fform_region_empty</code>	..Determines if the form contains elements...	12-91
<code>fform_replace_element</code>Replaces an element definition....	12-92
<code>fform_scroll_down</code>	..Scrolls the form down.....	12-93
<code>fform_scroll_left</code>	..Scrolls the form to the left.....	12-94
<code>fform_scroll_right</code>	..Scrolls the form to the right.....	12-95
<code>fform_scroll_up</code>Scrolls the form up.....	12-96
<code>fform_specify_group</code>	..Makes list of element tags into a group....	12-97
<code>fform_update</code>Draws/redraws the current form.....	12-98

Help Manager	13
General Description/Notes	13- 1
Defines	13- 2
<code>help_disable_high</code>	..Disables high and low level help.....	13- 3
<code>help_enable_high</code>	...Enables high and low level help.....	13- 4
<code>hlp_notify</code>Registers application notify routines.....	13- 5
<code>hlp_stop_notify</code>Turns off the low level help.....	13- 7
<code>ihm_add_state</code>Adds an application help state.....	13- 8
<code>ihm_delete_state</code>	...Deletes an application help state.....	13- 9
<code>ihm_new_entry</code>Adds a new entry to the help queue.....	13-10
<code>ihm_status</code>Returns the type of component user is in...	13-12

Information Box Manager	14
General Description/Notes	14- 1
Structures/Defines	14- 2
info_draw	Displays the information box on the screen...14- 5
info_draw_icon ...	Displays a draw list icon.....14- 6
info_put_string ..	Displays a word wrapped string.....14- 7
Keyboard Manager	15
General Description/Notes	15- 1
Defines	15- 3
kbd_flush	Clears the keyboard buffer.....15- 8
kbd_read	Reads a single CSR extended key code.....15- 9
kbd_scan	Scans the keyboard for an extended key code..15-10
Library Functions	16
General Description/Notes	16- 1
Structures/Defines	16- 2
about_versions	Displays the About dialog box.....16- 4
cnvrt_to_julian	Converts standard to julian format.....16- 5
cnvrt_to_std	Converts julian to standard format.....16- 6
compare_chars	Compares specified characters.....16- 7
compare_chars_ins ..	Compares case insensitive characters.....16- 8
dmmore_add_accessory ..	Adds an accessory to the More accessory..16- 9
dmmore_delete_accessory ..	Deletes an acc from the More acc.....16-11
get_ds	Gets the current data segment value...16-12
get_intl	Gets international structure information...16-13
get_month_strings ..	Gets the month abbreviations.....16-14
PageSetupDialog	Display DeskMate's Page Setup dialog box...16-15
str_upper	Converts a string to upper case.....16-16
to_upper	Converts a character to upper case.....16-17
waitloop	Waits for a specified period of time.....16-18

Menu Bar Manager	17
General Description/Notes	17- 1
Structures/Defines	17- 2
 mb_add_alarmAdds an alarm to the main menu bar.....	17- 6
mb_delete_alarm .Deletes an alarm from the menu bar.....	17- 7
mb_disableDisables auto processing of the menu bar.....	17- 8
mb_drawDraws the menu bar/registers menu accels.....	17- 9
mb_enableEnables auto processing of the menu bar.....	17-10
mb_eraseDeactivates the menu bar.....	17-11
mb_get_status ...Gets the number of the selected menu	17-12
 Message Box Manager	18
General Description/Notes	18- 1
Structures/Defines	18- 2
msg_drawDisplays a message box.....	18- 3
msg_runDisplays and processes a message box.....	18- 4
 Mouse Manager	19
General Description/Notes	19- 1
Structures/Defines	19- 2
ms_add_regionDefines mouse pointer to a region.....	19- 3
ms_default_pointerSpecifies default arrow pointer.....	19- 4
ms_define_pointerDefine a custom mouse pointer.....	19- 5
ms_delete_regionDeletes a mouse pointer region.....	19- 7
ms_disable_driverDisables the mouse driver.....	19- 8
ms_disable_regionsDeletes all mouse pointer regions.....	19- 9
ms_draw_pointerTurn on the mouse pointer.....	19-10
ms_enable_driverInitializes the mouse driver.....	19-11
ms_enable_regionsEnables all mouse pointer regions.....	19-12
ms_erase_pointerTurns off the mouse pointer.....	19-13
ms_get_statusGets the current mouse status.....	19-14

Information Box Manager	14
General Description/Notes	14- 1
Structures/Defines	14- 2
info_drawDisplays the information box on the screen...	14- 5
info_draw_icon ...Displays a draw list icon.....	14- 6
info_put_string ..Displays a word wrapped string.....	14- 7
 Keyboard Manager	 15
General Description/Notes	15- 1
Defines	15- 3
kbd_flushClears the keyboard buffer.....	15- 8
kbd_readReads a single CSR extended key code.....	15- 9
kbd_scanScans the keyboard for an extended key code..	15-10
 Library Functions	 16
General Description/Notes	16- 1
Structures/Defines	16- 2
about_versionsDisplays the About dialog box.....	16- 4
cnvrt_to_julianConverts standard to julian format.....	16- 5
cnvrt_to_stdConverts julian to standard format.....	16- 6
compare_charsCompares specified characters.....	16- 7
compare_chars_ins ..Compares case insensitive characters.....	16- 8
dmmore_add_accessory .Adds an accessory to the More accessory..	16- 9
dmmore_delete_accessory .Deletes an acc from the More acc.....	16-11
get_dsGets the current data segment value...	16-12
get_intlGets international structure information...	16-13
get_month_strings ..Gets the month abbreviations.....	16-14
PageSetupDialogDisplay DeskMate's Page Setup dialog box...	16-15
str_upperConverts a string to upper case.....	16-16
to_upperConverts a character to upper case.....	16-17
waitloopWaits for a specified period of time.....	16-18

Menu Bar Manager	17
General Description/Notes	17- 1
Structures/Defines	17- 2
 mb_add_alarmAdds an alarm to the main menu bar.....	17- 6
mb_delete_alarm .Deletes an alarm from the menu bar.....	17- 7
mb_disableDisables auto processing of the menu bar.....	17- 8
mb_drawDraws the menu bar/registers menu accels.....	17- 9
mb_enableEnables auto processing of the menu bar.....	17-10
mb_eraseDeactivates the menu bar.....	17-11
mb_get_status ...Gets the number of the selected menu	17-12
 Message Box Manager	18
General Description/Notes	18- 1
Structures/Defines	18- 2
msg_drawDisplays a message box.....	18- 3
msg_runDisplays and processes a message box.....	18- 4
 Mouse Manager	19
General Description/Notes	19- 1
Structures/Defines	19- 2
ms_add_regionDefines mouse pointer to a region.....	19- 3
ms_default_pointerSpecifies default arrow pointer.....	19- 4
ms_define_pointerDefine a custom mouse pointer.....	19- 5
ms_delete_regionDeletes a mouse pointer region.....	19- 7
ms_disable_driverDisables the mouse driver.....	19- 8
ms_disable_regionsDeletes all mouse pointer regions.....	19- 9
ms_draw_pointerTurn on the mouse pointer.....	19-10
ms_enable_driverInitializes the mouse driver.....	19-11
ms_enable_regionsEnables all mouse pointer regions.....	19-12
ms_erase_pointerTurns off the mouse pointer.....	19-13
ms_get_statusGets the current mouse status.....	19-14

Print Managers	20
General Description/Notes	20- 1
Structures/Defines	20- 3
Device Printing	
ptd_close	Closes the output device.....20- 6
ptd_draw_list	Adds a form to the page list.....20- 7
ptd_get_page	Gets the page setup and page mode.....20- 8
ptd_new_line	Adds a new (blank) line to the page list...20- 9
ptd_open	Prompts the user for a device to print to..20-10
ptd_preload	Loads the configuration printer driver.....20-12
ptd_print_far_page	Prints text strings from a far segment.....20-13
ptd_print_page	Prints the contents of the page list.....20-14
ptd_put_nchars	Adds a string to the page list.....20-15
ptd_put_nchars_x ...	ptd_put_nchars at a specified margin.....20-16
ptd_select	Selects the print device to be used.....20-17
ptd_set_page	Sets the page setup and page mode.....20-18
ptd_start_page	Initializes a new page.....20-19
Direct Printing	
prt_close	Closes the raw print device.....20-20
prt_get_printer	Gets printer driver information.....20-21
prt_open	Opens and initializes raw print device.....20-22
prt_put_char	Puts character to print device.....20-23
prt_put_nchars	Puts string to print device.....20-24
prt_put_tty	Puts non-filtered char to print device.....20-25
Sound Library	21
Introduction and Overview.....	21- 1
Hardware	21- 1
Software	21- 3
Structures/Defines	21- 5
snd_addbuf	Adds memory for the player to use.....21- 9
snd_amplify	Generates function table to amplify sound.21-10
snd_apply_func	Applies a function table to a sound.....21-11
snd_compensate	Creates a recording translation table.....21-12
snd_compress_part ...	Performs data compression on a sound.....21-13

snd_cue	Puts hardware in standby for play/record..	21-14
snd_decompress_part	.Performs data decompression on a sound....	21-15
snd_exit	Turns the sound chip off.....	21-16
snd_flush	Starts playing queued sounds.....	21-17
snd_init	Initializes the sound chip.....	21-18
snd_play	Plays sound(s) in memory.....	21-20
snd_record	Records a sound into memory.....	21-22
snd_stop	Stops the sound output or stops recording.	21-24
snd_version	Returns the version number.....	21-25
snd_volume	Sets the output level for play.....	21-26
snd_wait	Waits for sound(s) to finish.....	21-27
Appendix A	Compression/Decompression.....	21-28
DeskMate 88	DeskMate Sound 1988 compression.....	21-29
Appendix B	Bios Digital Sound Support.....	21-33
Appendix C	DeskMate Sound 1988 File Format.....	21-35
Appendix D	DeskMate Environment Issues.....	21-37

Spell Resource		22
General Description/Notes		22- 1
Structures/Defines		22- 2
spell_bind_end	Unloads spell resource.....	22- 3
spell_bind_init	Loads spell resource.....	22- 4
spl_add_cache_word	Adds a word to cache ram.....	22- 5
spl_add_user_word	Adds a word to the user dictionary.....	22- 6
spl_check_edit	Spell checks an editfield.....	22- 7
spl_check_text	Spell checks the TEXT application.....	22- 8
spl_check_word	Spell checks a word.....	22- 9
spl_delete_user_word ...	Deletes a word from user dictionary....	22-10
spl_get_alternates	Returns similarly spelled words.....	22-11
spl_init_cache	Re-initializes temporary word list.....	22-12
spl_init_user_dict	Clears user dictionary memory.....	22-13
spl_update_user_dict ...	Saves user dictionary to disk.....	22-14

Thesaurus Resource	23
General Description/Notes	23- 1
Structures/Defines	23- 3
thes_bind_endTerminates interface to the thesaurus.....	23- 4
thes_bind_initInitializes a thesaurus session.....	23- 5
thes_get_synGets synonyms for specified word.....	23- 6
thes_get_text_syn .Gets synonyms for DeskMate Text application.	23- 7
 Titleline Manager	 24
General Description/Notes	24- 1
Defines	24- 2
t11_disable_update ...Disables date/time updates.....	24- 3
t11_enable_updateEnables date/time updates.....	24- 4
t11_put_app_nameDisplays the application's name.....	24- 5
t11_put_data_nameDisplays the application's data name.....	24- 6
 Video Manager	 25
General Description/Notes	25- 1
Structures/Defines	25- 2
csr_load_video_driver .Loads a specified video driver.....	25- 8
fvid_draw_formDraws a form from far memory.....	25- 9
fvid_draw_form_element Draws a form element from far memory....	25-10
fvid_get_bounding_box .Gets a bounding box for a far form.....	25-11
vid_busy_disableDisables busy icon processing.....	25-12
vid_busy_enableEnables busy icon processing.....	25-13
vid_clear_blockClears specified block.....	25-14
vid_clear_screenClears current clip region.....	25-15
vid_clear_to_botClears to end of clip region.....	25-16
vid_clear_to_eolClears to end of current line.....	25-17
vid_dcx_to_wcxConverts device coordinate x.....	25-18
vid_dcy_to_wcyConverts device coordinate y.....	25-19
vid_delete_lineDeletes current line.....	25-20
vid_delete_ncharsDeletes specified number of characters..	25-21
vid_draw_arcDraws an arc graphic.....	25-22
vid_draw_bev_rectDraws a beveled edge rectangle.....	25-23

vid_draw_busy_icon	Draws the busy icon immediately.....	25-24
vid_draw_cursor	Draws the video cursor.....	25-25
vid_draw_ellipse	Draws an ellipse.....	25-26
vid_draw_form	Draws a form.....	25-27
vid_draw_form_element ..	Draws a form element.....	25-28
vid_draw_frame	Draws a frame.....	25-29
vid_draw_line	Draws a line.....	25-30
vid_draw_point	Draws a single pixel.....	25-31
vid_draw_polygon	Draws a polygon.....	25-32
vid_draw_polyline	Draws a series of connected line segs...	25-33
vid_draw_rect	Draws a rectangle.....	25-34
vid_draw_stroke	Draws a stroke character.....	25-35
vid_erase_cursor	Turns off the text cursor.....	25-36
vid_fill_block	Fills a rectangle area with pattern....	25-37
vid_get_attrs	Gets the current video attributes.....	25-38
vid_get_bounding_box ..	Gets a bounding box for a form element..	25-39
vid_get_buffer_size ...	Gets the minimum screen buffer size....	25-40
vid_get_clip	Gets the current clip region.....	25-42
vid_get_far_screen	Gets screen contents into a far buffer..	25-43
vid_get_palette	Gets current RGB value for palette.....	25-44
vid_get_screen	Gets the screen contents into a buffer..	25-45
vid_get_viewport	Gets the current viewport.....	25-47
vid_get_world	Gets current world boundaries.....	25-48
vid_inquire_colors	Fills VID_COLOR structure.....	25-49
vid_inquire_device	Fills VID_DEVICE structure.....	25-50
vid_insert_char	Inserts a character onto the screen....	25-51
vid_insert_line	Inserts a line onto the screen.....	25-52
vid_invert_block	Inverts fg and bg colors for a block....	25-53
vid_loadable_drivers ..	Returns the available video drivers....	25-54
vid_memory_free	Releases unused video memory.....	25-55
vid_memory_info	Returns info about unused vid memory...	25-56
vid_memory_inuse	Checks for unused video memory.....	25-57
vid_move_cursor	Moves cursor to specified location.....	25-58
vid_next_nwcx	Returns next device pixel x.....	25-59
vid_next_nwcy	Returns next device pixel y.....	25-60
vid_nextn_nwcx	Returns next n device pixel x.....	25-61

vid_nextn_nwcy	Returns next n device pixel y.....	25-62
vid_prev_nwcx	Returns prev device pixel x.....	25-63
vid_prev_nwcy	Returns prev device pixel y.....	25-64
vid_prevn_nwcx	Returns prev n device pixel x.....	25-65
vid_prevn_nwcy	Returns prev n device pixel y.....	25-66
vid_put_attr_nchars ...	Displays attribute character pairs.....	25-67
vid_put_char	Displays character and advances cursor..	25-68
vid_put_far_screen	Puts far buffer contents to screen.....	25-69
vid_put_image	Displays bitmap.....	25-70
vid_put_nchars	Displays string of specified length.....	25-72
vid_put_screen	Puts buffer contents to screen.....	25-73
vid_put_string	Displays null terminated string.....	25-74
vid_put_tty	Puts non-filtered string to screen.....	25-75
vid_read_cursor	Reads current cursor location.....	25-76
vid_restore_screen	Restores a portion of the screen.....	25-77
vid_save_screen	Saves a portion of the screen.....	25-78
vid_scroll_down	Scrolls screen down.....	25-79
vid_scroll_down_no_clear ..	Scrolls screen down w/o clearing ...	25-80
vid_scroll_left	Scrolls screen to the left.....	25-81
vid_scroll_left_no_clear ..	Scrolls screen left w/o clearing....	25-82
vid_scroll_right	Scrolls the screen to the right.....	25-83
vid_scroll_right_no_clear .	Scrolls screen right w/o clearing...	25-84
vid_scroll_up	Scrolls the screen up.....	25-85
vid_scroll_up_no_clear	Scrolls screen up w/o clearing.....	25-86
vid_set_attrs	Fills VID_ATTRS structure.....	25-87
vid_set_char_attr	Sets current attribute.....	25-88
vid_set_char_size	Sets size of system font.....	25-89
vid_set_clip	Sets the clip region.....	25-90
vid_set_clip_to_window .	Sets clip region to current window....	25-91
vid_set_colors	Sets fg and bg colors.....	25-92
vid_set_cursor_type	Sets current cursor type.....	25-93
vid_set_line_attr	Sets current line attributes.....	25-95
vid_set_palette	Sets the palette colors.....	25-96
vid_set_pattern	Sets current pattern.....	25-98
vid_set_viewport	Sets current viewport.....	25-99
vid_set_world	Sets current world boundaries.....	25-100

vid_wcx_to_dcxConverts world coordinate x.....25-101
vid_wcx_to_nwcxConverts world coordinate x to n.....25-102
vid_wcy_to_dcyConverts world coordinate y.....25-103
vid_wcy_to_nwcxConverts world coordinate y to n.....25-104

Window Manager26
General Description/Notes26- 1
Structure26- 2
win_activateMakes a specified window active.....26- 3
win_closeRemoves a window from the system.....26- 4
win_get_activeGets the currently active window.....26- 5
win_group_endDeletes base window and current group.....26- 6
win_group_initInitiates a new window group.....26- 7
win_openCreates and initializes a new window.....26- 8

About the *DeskMate Technical Reference 03.03.00*

The DeskMate Technical Reference is designed to assist programmers in the development of DeskMate applications. The manual is divided into twenty-six sections, listed in alphabetical order. The specific calls within the sections are also alphabetized making it easier to find specific call information. There are two types of sections, Resources and Managers. When a section name ends with the word "Resource", for example the Database Resource, an initialization call needs to be made before the functions may be used. When the section name ends with the word "Manager", for example the Video Manager, the section is part of a resource.

Managers listed as being part of the CSR (Core Services Resource) require the **csr_init** call be made before making any call in that manager. Managers listed as being part of the GUF (General User Functions) resource require the **guf_bind_init** or the **prguf_bind_init** call be made before making any call in that manager. The initialization calls need to be called only once and are described in detail in the Desk Executive section. If an application has called an initialization call it must call the corresponding terminate call before exiting, for example **csr_end** must be called if **csr_init** was called.

The **csr_init** initialization call must be made prior to calling any CSR routine. The **guf_bind_init** routine must be made prior to calling any GUF routine. The **prguf_bind_init** routine must be made prior to calling any PRGUF routine. The other initialization routines required by the separate resources, AUTOLOAD, DATABASE, FORM, SPELL, and THESAURUS, are described in their respective sections.

Autoload Resource

The Autoload Resource gives the applications the ability to pre-load required resources. The autoload resource has its own set of initialization and termination calls, **autoload_bind_init** and **autoload_bind_end** respectively. An example of the autoload resource's use is an application that needs to allocate space for a printer driver before it actually begins processing.

Clipboard Manager

The Clipboard Manager is a part of the Desk Executive and therefore requires no binding calls. It allows applications access to the DeskMate clipboard, so data may be transferred between the different applications.

Communication Manager

The Communication Manager, part of the CSR, requires the **csr_init** initialization call. The communication manager provides the applications with support for the different communication ports, COM1 and COM2.

Component Manager

The Component Manager, part of the CSR, requires the **csr_init** initialization call. The functions for all of the DeskMate components are described in the component manager. The components currently supported are: application defined components, check boxes, edit fields, icon buttons, list boxes, menu bars, push buttons, and radio buttons. Utilities for the different components can also be found in this section.

Configuration Manager

The Configuration Manager, part of the CSR, requires the **csr_init** initialization call. The configuration manager allows the programmer to retrieve, modify, and save the DeskMate configuration data stored in the DMCSR.CFG configuration data file.

Database Resources

The Database Resources contains the calls necessary to develop DeskMate database applications. The database resources provide indexed record level access into files both locally and on the network for applications. It will service one request at a time and will handle multiple users. The DeskMate database has several initialization calls, **db_bind_init** is used to bind to all pieces of the database. The **db_bind_read** initialization call binds to the database read resource. The **INIT_DB** initialization call binds to the update and build resources as requested by the application.

Desk Executive

The Desk Executive is neither a resource nor a manager, it is a small resident module which provides operating system services for DeskMate. It is responsible for the loading and releasing of resources. For this reason all of the Resource Initialization calls are listed in this section. The desk executive is also responsible for loading applications and accessories, managing the clipboard, and providing a communication system for task switching, and provides for some file/path searching routines.

Dialog Box Manager

The Dialog Box Manager, part of the CSR, requires the **csr_init** initialization call. The dialog box manager allows the application to prompt the user for several pieces of information simultaneously via a dialog box.

Environment Manager

The Environment Manager, part of the GUF, requires the **guf_bind_init** initialization call. The environment manager provides applications with a way to store application specific configuration information. The information may be used to return the application to a previous state since the information is saved to disk and retrieved whenever the application is subsequently run. The environment manager is similar in concept to the MS-DOS environment specified with "set" commands by the computer user.

Event Manager

The Event Manager, part of the CSR, requires the **csr_init** initialization call. The event manager returns events from both the keyboard and the pointing device. All events returned have a message type, a parameter, and a set of coordinates for mouse events. This is the preferable way to provide input into the application, so the application does not have to have separate processing for keyboard and the pointing device.

File I/O Manager

There are two different levels of file I/O support. The high level support is contained in the GUF resource requiring the **guf_bind_init** initialization call. The low level file I/O functions are contained in the PRGUF resource requiring the **prguf_bind_init** initialization call. High level file I/O provides block oriented support including user interface management. Low level file I/O provides the application direct file access.

Form Resources

The Form Resources are used to create pictures which can be printed, manipulated, or stored for later use. Each "form" is a list of the graphic and text primitives which make up a picture. The Form Resources have their own set of initialization and termination calls. To use the form resource as a *NEAR* resource the application should bind to the resource using **csr_form_bind_init** and terminate with **csr_form_bind_end**. If the application wishes to use the form resource as a *FAR* resource, the initialization calls are **fform_bind_init** and the termination call is **fform_bind_end**.

Help Manager

The Intelligent Help Manager contains all of the functions necessary for an application to provide context sensitive help for the user. The calls contained in this section are for use in the DeskMate 03.03.00 system only.

Information Box Manager

The Information Box Manager, part of the CSR, requires the `csr_init` initialization call. The information box manager supports the displaying of non-interactive messages on the screen. The information boxes support text strings, bitmaps, stroke fonts, and static boxes. Information boxes are very useful in demos and tutorials.

Keyboard Manager

The Keyboard Manager, part of the CSR, requires the `csr_init` initialization call. The keyboard manager makes direct CSR keyboard access available to the application and performs internal keyboard functions for the event manager.

Library Functions

The Library Functions section of the manual provides the application with many functions which are helpful to the developer but were not general enough to warrant their own sections. Most of these routines are contained in the two DeskMate libraries DM.LIB and DMMED.LIB and are linked in with the application that calls them. However some of these routines are contained in the GUF resource requiring the `guf_bind_init` initialization call, please make special note of these functions. The functions contained in this section of the manual include; the about dialog box, date conversion functions, international support functions, page setup dialog box, more accessory functions, and an application pausing function.

Menu Bar Manager

The Menu Bar Manager, part of the CSR, requires the `csr_init` initialization call. This section of the manual refers to the main application menu bar. Menu bars may also be run as components, refer to the component manager section for details on how to run the menu bar as a component.

Message Box Manager

The Message Box Manager, part of the CSR, requires the `csr_init` initialization call. A message box is a simplified dialog box, it contains a title, a static string message, and push buttons.

Mouse Manager

The Mouse Manager, part of the CSR, requires the `csr_init` initialization call. The mouse manager maintains and updates the mouse pointer for the application, as well as performing internal functions for the event manager. The application can control installation of the mouse drivers, the usage of the mouse pointer, definition of the mouse pointer, and mouse pointer regions on the display.

Print Managers

Most of the calls in the Print Managers section are a part of the CSR requiring the `csr_init` initialization call. The `ptd_open` call which prompts the user for the device to use, is part of the GUF resource requiring the `guf_bind_init` initialization call. There are two different types of printing support. The direct print support which prints directly to the printer port and the device printing support prints to any device; file, screen or printer.

Sound Library

The Tandy Digital Sound Library provides an API used to generate high quality sound for applications. The sound library makes use of the Digital to Analog Converter DAC in many Tandy computers.

Spell Resource

The Spell Resource is used by the spell checker accessory and by applications which wish to proof selected text. The spell engine resource is the actual engine and dictionary. The engine is loaded by the spell resource when the **spell_bind_init** initialization call is made.

Thesaurus Resource

The Thesaurus Resource is available to applications to provide a consistent user interface with an on-line synonym dictionary. The thesaurus resource provides the ability to locate synonyms for a word and optionally replace the word with a selected synonym to compose more readable documents. The thesaurus resource requires the **thes_bind_init** initialization call.

Titleline Manager

The Titleline Manager, part of the CSR, requires the **csr_init** initialization call. The titleline utilities manage the top line of the screen. These routines automatically display the date and time, the application name and application data file name.

Video Manager

The Video Manager, part of the CSR, requires the **csr_init** initialization call. The video manager provides the application all necessary direct video routines needed.

Window Manager

The Window Manager, part of the CSR, requires the **csr_init** initialization call. The window manager manages the creation, deletion, and activation of application defined windows.

Autoload Resource

"Autoload Resource"

Table of Contents

General Description/Notes	1- 1
Structures/Defines	1- 2
autoload_bind_end ...Release the autoload resource.....	1- 4
autoload_bind_init ..Initializes autoload resource.....	1- 5
autoload_setRegister and Delete autoload resource.....	1- 6

DeskMate Technical Reference Autoload Resource

General Description/Notes

The autoloader enables an application to have the desk executive pre-load a resource it will need before it is executed.

The Autoload resource is not part of the DeskMate Runtime System, if you need this resource please contact Tandy DeskMate Development Services.

The application calls the autoload passing it needed information such as the name of the resource to load, initialization information, and the resource load priority. Autoloader stores this information in a file and the desk executive uses autoloader to load requested resources when desk is run. When loaded, the resource is called with the stored initialization information in order to start or just set up the resource. After loaded the DESK executive uses the load priority defines to decide when the resource is loaded and unloaded.

The Spell Accessory uses the autoloader for auto-proofing.

DeskMate Technical Reference

Autoload Resource

Structures/Defines

The following structure must be initialized and passed to autoloader to register a resource to be autoloaded:

```

/*-----*/
/*      AUTOLOAD.H - defines and structures used by      */
/*      applications calling the autoload resource.      */
/*-----*/

struct autoload_struct
{
    char  CallerId;    /* define code for application that */
                    /* requested autoload for resource */
    char  ResourceName[9]; /* ASCIIZ string containing the */
                    /* name of the resource load (all */
                    /* capitals) */
    char  FunctionId;   /* number of function to call in */
                    /* resource for it to initialize or */
                    /* start running */
    char  LoadPriority; /* define code for the resource load */
                    /* priority */
    char  FunctionParams[5]; /* a list of parameters the */
                    /* initialization function needs */
}
typedef struct autoload_struct AUTOLOAD;

```

CallerId is a code assigned to an application that requests the autoload resource. Each current application receives a unique CallerID.

ResourceName is a 9-byte ASCIIZ string which contains the name of a resource requested by an application.

FunctionId identifies the particular function within a given resource. The FunctionID value is used to initialize or start the function.

```

/* define for FunctionId */
NO_FUNCTION      -1    /* define for FunctionId if there is no */
                    /* initialization function */

AUTO_SPELL       1     /* spell checker CallerId */

```

When the function is called for initialization the FunctionId is put in AX and the FunctionParams list is pointed to by ES:BX. The parameters can contain any information as long as it is 5 bytes long. If the application does not need to be initialized then the FunctionId should equal NO_FUNCTION.

LoadPriority establishes the load priority of one resource with respect to all other resources requested. The values defined for LoadPriority are described below. When the desk executive needs more memory, no one application can be using the resource for it to be unloaded.

```

/* defines for LoadPriority */
LOAD_PRIORITY0   0     /* resource is loaded at applications */
                    /* request and freed when application is */
                    /* exited (removes resource from autoload) */
LOAD_PRIORITY1   1     /* resource is loaded at when the desk */
                    /* executive starts and unloaded for */
                    /* non-DeskMate applications and reloaded */
                    /* after non-DeskMate applications exit and */
                    /* freed upon exit from the desk executive */

```

DeskMate Technical Reference

Autoload Resource

LOAD_PRIORITY2	2	/* resource is loaded when the desk executive */ /* starts and stays in until exit from */ /* the desk executive */
LOAD_PRIORITY3	3	/* resource is loaded before the DESK */ /* executive and remains after the DESK */ /* executive is exited (preload resource) */

FunctionParams is a 5-byte list of parameters needed by the initialization function; the format of the parameter string is determined by the resource and function that will use the parameter information.

DeskMate Technical Reference

Autoload Resource

autoload_bind_end ()

autoload_bind_end enables applications to release the autoload resource.

Input Parameters

None.

Return Value

None.

Special Notes

If an application calls **autoload_bind_init**, it must call **autoload_bind_end** when finished with the autoload resource.

DeskMate Technical Reference

Autoload Resource

autoload_bind_init ()

autoload_bind_init sets up the bindings for the autoload resource, enabling the application to use functions in the autoloader.

Input Parameters

None.

Return Value

<int>

DM_ERROR if **autoload_bind_init** was unsuccessful.

Special Notes

autoload_bind_init should be made before calling any autoloader calls.

DeskMate Technical Reference

Autoload Resource

autoload_set (pAUTOLOAD)
AUTOLOAD *pAUTOLOAD;

autoload_set maintains the file containing the resources load priority settings. **autoload_set** is used to register new entries in the file or to delete existing entries from the file.

Input Parameters

pAUTOLOAD is a pointer to an AUTOLOAD structure that contains the resource information.

Return Value

<int>

DM_ERROR if **autoload_set** was not able to find or create the autoload configuration file.

Special Notes

In order to delete an entry in the autoload file the application should send a structure which contains LOAD_PRIORITY0 in the LoadPriority element. There can be multiple entries for the same resource as long as the CallerId elements are different in the AUTOLOAD structure. The highest priority found for a resource will be the one stored by the desk executive. If no initialization call is needed then the FunctionId should equal NO_FUNCTION.

Clipboard Manager

"Clipboard Manager"

Table of Contents

General Description/Notes	2- 1
Defines	2- 2
 dm_clear_clipboard	 Clears data from Clipboard..... 2- 3
dm_clipboard_size	Returns size of Clipboard memory..... 2- 4
dm_free_clip_buffer	Returns memory to Clipboard..... 2- 5
dm_get_clip_buffer	Gets unused space in Clipboard memory. 2- 6
dm_get_clipboard_info ...	Gets type, length, & pointer..... 2- 7
dm_set_clipboard_info ...	Sets type, length..... 2- 8

DeskMate Technical Reference Clipboard Manager

General Description/Notes

The clipboard manager is a part of the Desk Executive.

Refer to the DeskMate Development Guide Programming Examples Special Topics for examples of how to read from and write to the clipboard.

DeskMate Technical Reference

Clipboard Manager

Defines

```
/*-----*/
/*
/* DMEEXEC.H contains the Clipboard defines */
/*
/*-----*/

DM_OK          1      /* value returned when executive request */
                  /* was succesful */
DM_ERROR       -1     /* value returned when an error in */
                  /* executive request occured */

/* These are the different clipboard type */
/* available to the applications. */
CLIP_EMPTY     0
CLIP_TEXT      1
CLIP_BITMAP    2
CLIP_WORKSHEET 3
CLIP_FILER     4
CLIP_PAINT     5
CLIP_DRAW      6
CLIP_DRAW TEXT 7
CLIP_MUSIC     8
CLIP_CALENDAR  9
```

DeskMate Technical Reference

Clipboard Manager

dm_clear_clipboard ()

dm_clear_clipboard clears the clipboard of any data, and sets the clipboard type to CLIP_EMPTY and the clipboard length to zero.

Input Parameters

None.

Return Value

None.

DeskMate Technical Reference Clipboard Manager

dm_clipboard_size ()

dm_clipboard_size returns the size of the clipboard memory in bytes.

Input Parameters

None.

Return Value

The size of the clipboard in bytes.

Special Notes

dm_clipboard_size is only available when running DeskMate versions 03.03 and later.

The return value is the maximum size that can be stored in the clipboard. If the clipboard buffer is in use its size is not included in the size returned by **dm_clipboard_size**.

DeskMate Technical Reference

Clipboard Manager

dm_free_clip_buffer ()

dm_free_clip_buffer returns the memory to the clipboard which was received from **dm_get_clip_buffer**.

Input Parameters

None.

Return Value

None.

Special Notes

dm_free_clip_buffer is only available when running DeskMate 03.03 and later.

DeskMate Technical Reference

Clipboard Manager

dm_get_clip_buffer (lpLength, lpBuffer)
int far *lpLength;
char far **lpBuffer;

dm_get_clip_buffer gets the length of and pointer to unused space in the clipboard memory.

Input Parameters

lpLength is a far pointer to the integer in which to store the length of the clipboard data.
lpBuffer is a far pointer to the application's long pointer variable.

Return Value

NULL if no room is left on the clipboard, or the buffer is already in use.
DM_OK if the buffer is returned.

Special Notes

dm_get_clip_buffer is only available when running DeskMate 03.03 and later.

While the clipboard buffer is in use (until **dm_free_clip_buffer** is called), the size of the clipboard is reduced to the size of its contents at the time **dm_get_clip_buffer** was called. The desk executive does not automatically free the clipboard buffer when a task exits, but will free it whenever there are no tasks running.

See Also

dm_free_clip_buffer()

DeskMate Technical Reference Clipboard Manager

dm_get_clipboard_info (lpType, lpLength, lpBuffer)

int far *lpType;

int far *lpLength;

char far **lpBuffer; *← This is wrong. Should be: char far * far *lpBuffer*

dm_get_clipboard_info returns information about the current clipboard contents (the type and length of the data, and a far pointer to the clipboard buffer).

Input Parameters

lpType is a far pointer to the integer in which to store the clipboard data type.

lpLength is a far pointer to the integer in which to store the length of the clipboard data.

lpBuffer is a far pointer to the application's long pointer variable.

Return Value

The information is returned in the parameters.

Special Notes

See Chapter 7 of the MicroSoft "C" manual for more help on far pointers to far pointers.

The following are the current clipboard data types:

CLIP_EMPTY	0
CLIP_TEXT	1
CLIP_BITMAP	2
CLIP_WORKSHEET	3
CLIP_FILER	4
CLIP_PAINT	5
CLIP_DRAW	6
CLIP_DRAW_TEXT	7
CLIP_MUSIC	8
CLIP_CALENDAR	9

Example

Check_Clipboard()

```
{  
    int    Type, Length;  
    char far *lpBuffer;  
  
    dm_get_clipboard_info( (int far *)&Type, (int far *)&Length,  
                           (char far **)&lpBuffer );  
    .  
    .  
    .  
}
```

Should be:

*(char far * far *)*

DeskMate Technical Reference Clipboard Manager

dm_set_clipboard_info (Type, Length)

int Type;

int Length;

dm_set_clipboard_info sets the clipboard data type and length.

Input Parameters

Type is the data type of the data to be put into the clipboard buffer.

Length is the number of bytes to be put into the clipboard buffer.

Return Value

<int>

DM_OK if clipboard buffer is large enough for the requested size.

DM_ERROR if requested size is too large.

Communications Manager

"Communication Manager"

Table of Contents

General Description/Notes	3- 1
Structures/Defines	3- 2
com_call	Calls the specified number..... 3- 4
com_close	Closes the current device..... 3- 5
com_disc	Disconnects the terminal session..... 3- 6
com_get_buffer_info ...	Gets com structure information..... 3- 7
com_get_lines_states ..	Gets the com port status..... 3- 8
com_get_parms	Gets the communication parameters..... 3- 9
com_open	Opens the communication device..... 3-10
com_put_char	Writes a character to the com device.... 3-11
com_put_string	Writes a string to the com device..... 3-12
com_send_break	Sends break sequence to com device..... 3-13
com_set_parms	Sets the communications parameters..... 3-14
com_test_carrier	Check com port for carrier signal..... 3-15

DeskMate Technical Reference Communications Manager

General Description/Notes

The functions in the Communications Manager provide the ability to perform communications. These routines allow the application to set up communications parameters, open and close the communications ports and dial and hang up the phone. The application can either send one character at a time or an ASCII string to the communications port. Functions are also provided for detecting the carrier and sending a break sequence. The **com_open** call allocates a buffer for a serial interrupt information about this buffer can be obtained with the **com_get_buf_info** call.

The user must use the Setup Accessory to initialize the COM ports before calling **com_open**; if not, **com_open** will notify the application.

DeskMate Technical Reference Communications Manager

Structures/Defines

```
/*-----*/
/*
/* CSRCFG.H contains the Communications structures and defines */
/*
/*-----*/

/* Communications */
struct cparms_defn
{
    char    bXon;           /* Xon/Xoff enable flag */
    char    bASCFfilter;    /* ASCII filter enable flag */
    char    bLFFfilter;     /* line filter enable flag */
    char    bStatus;        /* com port status flag (OPEN or CLOSED) */
    char    bCarrier;       /* carrier det stat flag (CARRY/NO_CARRY) */
    int     baud_rate;      /* line speed setting */
    char    word_size;      /* 7 or 8 */
    char    parity;         /* EVEN PARITY, ODD PARITY or NO PARITY */
    char    stop_bits;      /* 1 or 2 */
};
typedef struct cparms_defn CPARMS;
```

CPARMS.bXon:

Xon/Xoff enable flag (ENABLED or DISABLED).

CPARMS.bASCFfilter:

ASCII filter enable flag (ENABLED or DISABLED).

CPARMS.bLFFfilter:

Line filter enable flag (ENABLED or DISABLED).

CPARMS.bStatus:

Com port status flag.

/* Port status */

PORT_OPEN 1

/* terminal active - dtr and rts */

PORT_CLOSED 0

/* terminal not ready - no dtr or rts */

CPARMS.bCarrier:

Carrier detect status flag.

/* Carrier state */

CARRY 1

NO_CARRY 0

CPARMS.baud_rate:

Line speed setting.

/* int value of 300, 1200, 2400, 4800, 9600 */

CPARMS.word_size:

7 or 8.

CPARMS.parity:

/* Parity settings */

NO PARITY 0

ODD PARITY 1

EVEN PARITY 2

DeskMate Technical Reference Communications Manager

CPARMS.stop_bits:
1 or 2.

```
struct com_buffer_info_defn
{
    char far *rs232_start_ptr;
    char far *rs232_end_ptr;
    char far *input_ptr;
    char far *output_ptr;
    int      rs232_buf_size;
    int      count_num;
    char      xon_xoff_flag;
};
typedef struct com_buffer_info_defn COM_BUFFER_INFO;
```

```
struct com_state_defn
{
    char      com_port;      /* COM1 or COM2 */
    char      line_type1;    /* DIRECT or MODEM */
    char      modem1;        /* modem name index */
    char      line_type2;    /* DIRECT or MODEM */
    char      modem2;        /* modem name index */
    char      line_type3;    /* DIRECT or MODEM */
    char      modem3;        /* modem name index */
    char      line_type4;    /* DIRECT or MODEM */
    char      modem4;        /* modem name index */
};
typedef struct com_state_defn COM_STATE;
```

```
/* Port addresses */
COM1      0
COM2      1
COM3      2
COM4      3
```

```
/* Line types */
MODEM      1      /* modem is attached to line */
DIRECT     0      /* direct connect line */
```

The currently supported modems are:

HAYES, PLUS 1200, INTERNAL 1200, PLUS 300 HAYES, DCM 212 HAYES, DCM 7 HAYES, DCM 5, DC 2212, DC 1200, INTERNAL 300, and MODEM II.

DeskMate Technical Reference Communications Manager

com_call (pString, Type)

char *pString;

int Type;

com_call dials the specified phone number.

Input Parameters

pString is a pointer to a null terminated ascii string containing the phone number to call.

Type describes how to dial the number. 0 = auto-execution, 1 = dial only.

Return Value

<int>

One (1) if the call was successful.

CSR_NULL if there is no carrier.

CSR_ERROR if an error occurs during the call.

ESC_KEY if the ESC key was pressed during the call.

Special Notes

An error can occur under any of the following conditions:

- The device is in the PORT_OPEN state and the call cannot open the device.

- There is no RS232 installed.

- The communication lines are direct-connect.

- The communication timeout delay is reached.

The user may press the escape key to cancel the calling process.

If the **type** is defined to be auto-execution then full processing of the defined autolog file occurs.

If the **type** is defined to be dial only then the only processing which takes place is the dialing of the phone, this is normally used when voice connection is desired.

DeskMate Technical Reference Communications Manager

com_close (pCPARMS)
CPARMS *pCPARMS;

com_close closes and disconnects the current communications device, resets the device status, and drops dtr (data terminal ready) and rts (request to send).

Input Parameters

pCPARMS is a pointer to a communications parameters structure.

Return Value

<int>

CSR_NULL if successful.

CSR_ERROR if the com port was not open.

Special Notes

The bytesize may be 7 or 8.

The parity may be NO_PARITY, ODD_PARITY, or EVEN_PARITY.

The stopbits may be 1 or 2.

Xon_Xoff may be ENABLED or DISABLED.

The ascii filter may be ENABLED or DISABLED.

The line feed filter may be ENABLED or DISABLED.

The carrier detect may be CARRY or NO_CARRY.

See Also

The CPARMS structure defined at the beginning of this chapter.

DeskMate Technical Reference Communications Manager

com_disc ()

com_disc disconnects the terminal connection.

Input Parameters

None.

Return Value

None.

DeskMate Technical Reference Communications Manager

com_get_buffer_info (plpCOM_BUFFER_INFO)
COM_BUF_INFOR far **plpCOM_BUFFER_INFO;

com_get_buffer_info returns the far address of the COM_BUF_INFOR structure. This is the serial interrupt structure which was initialized by **com_open**.

Input Parameters

plpCOM_BUF_INFOR is a far pointer to a far COM_BUFFER_INFO pointer.

Return Value

The buffer is modified with the far COM_BUFFER_INFO pointer.

Special Notes

The buffer information structure is included in CSRCFG.H.

Example

```
COM_BUF_INFOR far *lpComBuf;  
.  
.  
com_get_buffer_info( &lpComBuf );
```

See Also

com_open()

DeskMate Technical Reference Communications Manager

com_get_line_states ()

com_get_line_states returns the status of the communications ports.

Input Parameters

None.

Return Value

<int>

CSR_NULL if no lines are being used.

The low 4 bits of this value are used to indicate which ports are in use. The least significant bit corresponds to the COM1 state, the next, COM2, etc. A 0 indicates an inactive port, a 1, active. No more than two bits will be on at one time.

DeskMate Technical Reference Communications Manager

com_get_parms (pCPARMS)
CPARMS *pCPARMS;

com_get_parms returns the current communications parameters.

Input Parameters

pCPARMS is a pointer to a communications parameters structure.

Return Value

The current parameters are returned in the structure.

Special Notes

The bytesize may be 7 or 8.
The parity may be NO_PARITY, ODD_PARITY, or EVEN_PARITY.
The stopbits may be 1 or 2.
Xon_Xoff may be ENABLED or DISABLED.
The ascii filter may be ENABLED or DISABLED.
The line feed filter may be ENABLED or DISABLED.
The device status may be PORT_OPEN or PORT_CLOSED.
The carrier detect may be CARRY or NO_CARRY.

DeskMate Technical Reference Communications Manager

com_open (pCPARMS)
CPARMS * pCPARMS;

com_open opens a communications device, sets the device status, and invokes dtr (data terminal ready) and rts (request to send).

Input Parameters

pCPARMS is a pointer to a communications parameters structure.

Return Value

<int>

CSR_NULL if the device was opened.

CSR_ERROR if there was an error opening the device.

CSR_DEFAULT if the communications accessory has not yet been run, to allow access to the COM ports.

Special Notes

com_open allocates 4K (or as much as possible if 4K is not available) for an RS232 serial interrupt buffer. It then sets the buffer information so that it may be accessed with **com_get_buf_info**.

The bytesize may be 7 or 8.

The parity may be NO_PARITY, ODD_PARITY, or EVEN_PARITY.

The stopbits may be 1 or 2.

Xon_Xoff may be ENABLED or DISABLED.

The ascii filter may be ENABLED or DISABLED.

The line feed filter may be ENABLED or DISABLED.

The device status must be PORT_CLOSED.

The carrier detect may be CARRY or NO_CARRY.

DeskMate Technical Reference Communications Manager

com_put_char (Character)

char Character;

com_put_char writes a character to the communications device.

Input Parameters

Character is the character to write to the device.

Return Value

<int>

CSR_NULL if the write was successful.

CSR_ERROR if an error occurred while writing to the device.

DeskMate Technical Reference Communications Manager

com_put_string (pString)
char *pString;

com_put_string sends a null terminated string to the I/O port.

Input Parameters

pString is a pointer to a null terminated string.

Return Value

<int>

CSR_ERROR if an error occurred.

ESC_KEY if the user pressed the ESC_KEY to cancel the function.

CSR_NULL if the string is sent successfully.

DeskMate Technical Reference Communications Manager

com_send_break ()

com_send_break sends a break sequence to the communication port.

Input Parameters

None.

Return Value

None.

DeskMate Technical Reference Communications Manager

com_set_parms (pCPARMS)
CPARMS *pCPARMS;

com_set_parms sets the current communications parameters.

Input Parameters

pCPARMS is a pointer to a communications parameters structure.

Return Value

None.

Special Notes

The bytesize may be 7 or 8.

The parity may be NO_PARITY, ODD_PARITY, or EVEN_PARITY.

The stopbits may be 1 or 2.

Xon_Xoff may be ENABLED or DISABLED.

The ascii filter may be ENABLED or DISABLED.

The line feed filter may be ENABLED or DISABLED.

DeskMate Technical Reference Communications Manager

com_test_carrier ()

com_test_carrier checks the communications port for a carrier signal.

Input Parameters

None.

Return Value

<int>

CARRY if there is a carrier detected.

CSR_NULL if no carrier is detected.

Component Manager

"Component Manager"

Table of Contents

General Description/Notes	4- 1
Structures/Defines	4- 2
Application Defined Components	
General Description/Notes	4-15
Structures/Defines	4-16
cmp_addAdds an application defined component.....	4-17
cmp_closeCloses an application defined component.....	4-19
cmp_deleteDeletes an application defined component.....	4-20
cmp_drawDraws an application defined component.....	4-21
cmp_openOpens an application defined component.....	4-22
cmp_runRuns an application defined component.....	4-23
Pre-Defined Components	
General Description/Notes	4-25
Check box	4-25
Edit field	4-25
Icon button	4-26
List box	4-26
Menu bar	4-26
Push button	4-26
Radio buttons	4-27
Edit field examples	
Single line editfield	4-28
Formatted editfield	4-29
Multi line editfield	4-30
cmp_closeCloses a Pre-Defined component.....	4-31
cmp_drawDraws a Pre-Defined component.....	4-32
cmp_openOpens a Pre-Defined component.....	4-33
cmp_runRuns a Pre-Defined component.....	4-34

Component Utilities

edt_clear	Clears selected edit field text.....	4-36
edt_copy	Copies edit field text to clipboard.....	4-37
edt_cut	Cuts edit field text to clipboard.....	4-38
edt_format_multi	.Formats edit field string for wordwrap.....	4-39
edt_form_draw	Adds edit field lines to form.....	4-40
edt_paste	Pastes clipboard to edit field.....	4-41
edt_replace	Replaces edit field text w/specified text....	4-42
lb_sort_list	Sorts the items within a list box.....	4-43

DeskMate Technical Reference Component Manager

General Description/Notes

The component manager is a common interface to all components, CSR and application defined. All components must be processed through the component manager.

The component manager processes components for the application. This also allows an application to define a component and use it consistently with CSR-defined components.

Because all components are processed by the component manager, each component structure must begin with a common header. Refer to "Structures/Defines" for a description of the `CMP_HEADER` structure that is represented by the 'header' element of each component structure.

The CSR supports the following components:

- application defined components
- check boxes
- edit fields
- icon buttons
- list boxes
- menu bars
- push buttons
- radio buttons
- scroll bars

The scroll bar is a user-defined component which may be linked with the calling application. To use the scroll bar, first, include `DMSCROLL.H` or `DMSCROLL.INC` and define a `SCROLL_BAR` structure tailored for your needs. Use the default values provided in the include file when defining your structure. Then, call `add_scroll_bar_cmp` and store the return value, if not `CSR_ERROR`, in the `type` element of the `CMP_HEADER` in the `SCROLL_BAR` structure. Link with the appropriate DeskMate library. To delete the scroll bar, call `cmp_delete` with the return value from the previously called `add_scroll_bar_cmp`, if not `CSR_ERROR`.

The edit field provides for selection of data within the edit window. The application must decide if it wishes to allow this selected data to be "cut" or "copy-ed" to the clipboard. When the edit field returns control to the application, the `select_offset` and `select_length` will remain in the structure as they were in the edit field. The application must decide by checking the `select_length` if the "cut" and "copy" menu items should be available. The application needs to check the `select_length` ANY time the edit field returns to the application. If those menu items are available, and the user subsequently selects a clipboard menu item, the application must call one of the following routines to allow the edit field to perform the operation.

- `edt_clear`
- `edt_copy`
- `edt_cut`
- `edt_format_multi`
- `edt_form_draw`
- `edt_paste`
- `edt_replace`

`lb_sort_list` is called when sorting of the items within a list box is desired, without running or drawing the list box.

DeskMate Technical Reference Component Manager

Typically, the CSR will prevent all components from receiving mouse hold events except for the window in which the button down event was detected. This is accomplished within the CSR by recording the handle of the current window when a button down is detected. All subsequent mouse hold events are returned only if the currently active window has that handle.

This allows two different components to receive all mouse hold events. This occurs when one component receives a button down event, then completes running or drawing, and then another component is run or drawn. This occurs because the handle is freed when the first component is closed, making it available for the next created component. This problem will only occur if a component is created, reads a button down event, is then closed prior to the termination of that hold sequence, and a new window is then created that processes events.

DeskMate Technical Reference Component Manager

Structures/Defines

```
/*-----*/
/*
/* CSRCMPS.H contains the Components structures and defines */
/*
/*-----*/
```

All components, whether defined by the application or by the CSR, are processed by the component manager. The CMP_HEADER structure for a component defines its type, location, size, accelerator, and return value.

```
/* General */
struct cmp_header_defn
{
    unsigned int  type;           /* component type */
    char          bEnabled;       /* enable flag for component */
    MAPRECT       maprect;        /* origin/extent */
    unsigned int  accel;          /* key accelerator for component */
    unsigned int  return_code;    /* return code for component */
};
typedef struct cmp_header_defn CMP_HEADER;
```

type:

The type of component as defined below, or as returned by cmp_add.

```
/* Types */
CMP_PUSHBUTTON      CMP_TAG+0
CMP_RADIOBUTTONS    CMP_TAG+1
CMP_ICONBUTTON      CMP_TAG+2
CMP_LISTBOX         CMP_TAG+3
CMP_CHECKBOX        CMP_TAG+4
CMP_EDIT            CMP_TAG+5
CMP_MENUBAR         CMP_TAG+6
```

bEnabled:

The enable (grayed) flag for component (ENABLED, DISABLED or CMP_NO_TAB).

```
/* Keep component out of tab order */
CMP_NO_TAB          0x02
```

maprect:

The location and size of the component.

```
MAPRECT.xorg        int - world coordinate x origin of the component
                        hot spot and working window.
MAPRECT.yorg        int - world coordinate y origin of the component
                        hot spot and working window.
MAPRECT.xext        int - world unit x extent of the component hot
                        spot and working window.
MAPRECT.yext        int - world unit y extent of the component hot
                        spot and working window.
```

accel:

The keyboard accelerator for the component hot spot.

```
NO_ACCEL            -1      /* no accelerator */
```

return code:

The return code for the component hot spot.

```
/* Return codes */
CMP_NO_ACTION       0      /* no action taken within component */
CMP_ACTION          1      /* component status has changed */
CMP_SCROLLED        2      /* scroll occurred */
CMP_TRUNCATED       3      /* edit string was truncated */
CMP_CANCEL          5      /* component was canceled */
```

DeskMate Technical Reference Component Manager

```

CMP_DISABLED      6          /* component was grayed */
CMP_GO            7          /* user invoked select and go */
CMP_SELECT_CHANGE 8          /* user invoked selection change */
CMP_ACTION_IN_EVENT 9        /* cmp action is in the event queue */
CMP_APPL          10         /* cmp received an EVENT_APPL event */

```

```

/* No handle */
CMP_NULL_HANDLE   -1

```

```

/* Null states */
NO_SELECTED      -1          /* no selected */

```

The push button structure is defined as follows:

```

/* Push button Component */
struct pushbutton_defn
{
    CMP_HEADER      header;
    char            type;          /* flat or raised */
    char            bState;        /* up/down flag */
    char            pattern;       /* pattern number of background */
    unsigned char   *pString;      /* pointer to button name string */
};
typedef struct pushbutton_defn PUSHBUTTON;

```

Pushbutton.header:

This header is a header structure that provides information to the component manager such as where the push button is to be displayed. Refer to the CMP_HEADER documentation for a further discussion of component headers.

```

/* Header maprect */
PB_BASE_XEXT      2*CHAR_XEXT

PB_YEXT           345 - Used to define the y extent of all push buttons.
PB_DEFAULT_ACCEL  CSR_DEFAULT
x * CHAR_XEXT + PB_BASE_XEXT, Where x is the number of character in the string label.

```

Pushbutton.type:

Indicates whether the push button is a two dimensional (flat) or a three dimensional (raised) push button while in the non-pressed state. It also indicates whether the push button is to be inverted when pressed, if it is a raised push button.

```

PB_FLAT          1          /* flat push button */
PB_RAISED        0          /* raised push button */
PB_INV_RAISED    2          /* invert raised push button on press */

```

Pushbutton.bState:

Indicates whether the push button is selected or not.

```

PB_UP            DESELECTED
PB_DOWN          SELECTED

```

Pushbutton.pattern:

Defines pattern number of the background upon which the push button is drawn. This option is used with a raised push button to restore the background of the area the push button covered before it was pressed.

DeskMate Technical Reference Component Manager

PATTERN1 - PATTERN20.

Pushbutton.pString:
Points to the button label string.

DeskMate Technical Reference Component Manager

Extent constants:

PB_BASE_XEXT - Used to calculate the x extent of the push button. Use the following formula to define the x extent of a push button: $PB_BASE_XEXT + (n * CHAR_XEXT)$, where 'n' is the number of characters in the label string.

/* Radio buttons Component */

Radio button groups use two kinds of structures: one for the entire radio button group, another for each individual button in the group.

```
struct radiobutton_defn
{
    char    bEnabled;    /* enable (grayed) flag */
    int     xorg;        /* x origin relative to group */
    int     yorg;        /* y origin relative to group */
};
typedef struct radiobutton_defn RADIOBUTTON;
```

Radiobutton.bEnabled

Enable flag for the radio button (ENABLED or DISABLED).

Radiobutton.xorg:

World coordinate x origin relative to the radio buttons group origin.

Radiobutton.yorg:

World coordinate y origin relative to the radio buttons group origin.

```
struct rb_group_defn
{
    CMP_HEADER header;
    char       nAcross;    /* number of buttons wide */
    char       nDown;      /* number of buttons tall */
    char       selected;    /* ordinal id of pushed button */
    char       pattern;     /* pattern number of background */
    char       nButtons;    /* number of buttons in group */
    RADIOBUTTON *pButtons; /* pointer to RADIO BUTTON structures */
    char       cursor;     /* cursor button index */
};
typedef struct rb_group_defn RB_GROUP;
```

Radiobutton.header:

This header is a header structure which provides information to the component manager such as where the radio button group will be displayed. Refer to the CMP_HEADER documentation for a further discussion of component headers.

Radiobutton.nAcross:

Number of radio buttons from left to right in the radio buttons group.

Radiobutton.nDown:

Number of radio buttons from top to bottom in the radio buttons group.

Radiobutton.selected:

Index of the currently pressed radio button. The index is zero relative and refers to the radio buttons in the order that they are defined as pointed to by pButtons.

Radiobutton.pattern:

*Radio buttons are
listed in column-
major order.*

DeskMate Technical Reference Component Manager

Pattern number of the background upon which the radio buttons are drawn (PATTERN1 to PATTERN20). This option is used with raised radio button to restore the background of the area the radio button covered before it was pressed.

Radiobutton.nButtons:

Number of radio buttons in the group.

Radiobutton.pButtons:

Pointer to the RADIOBUTTON structures.

Radiobutton.cursor:

0 to nButtons-1. CSR_DEFAULT defines where the cursor is placed when the group is run. Use the value of the desired button in the group.

```
/* Radio button extents */
RB_XEXT      3*CHAR_XEXT
RB_YEXT      CHAR_YEXT

/* Icon button Component */
struct iconbutton_defn
{
    CMP_HEADER header;
    char type; /* type of icon button */
    char bState; /* on/off flag */
    char pattern; /* pattern number of background */
    char interior; /* interior pattern of button */
    char *pUp; /* icon "on" definition */
    char *pDown; /* icon "off" definition */
};
typedef struct iconbutton_defn ICONBUTTON;
```

ICONBUTTON.header:

Header is a header structure that provides information to the component manager such as where the icon button will be displayed. Refer to the CMP_HEADER documentation for a further discussion of component headers.

ICONBUTTON.type:

Indicates whether the icon button is a two dimensional (flat) or a three dimensional (raised) icon button while in the deselected, non-pressed state. It also indicates whether the icon button is to be inverted when pressed, if it is a raised icon button.

Icon button types:

IB_RAISED	0 - three dimensional, raised icon button.
IB_FLAT	1 - two dimensional, flat icon button.
IB_INV_RAISED	2 - three dimensional, raised icon button that is inverted when pressed.
IB_NO_BORDER	3 - flat button with no border drawn around the icon.

ICONBUTTON.bState:

Indicates whether the icon button is pressed or not (BUTTON_UP or BUTTON_DOWN).

ICONBUTTON.pattern:

Pattern number of the background upon which the icon button is drawn (PATTERN1 to PATTERN20). This option is used to restore the background of the area the icon button covered before it was pressed.

DeskMate Technical Reference Component Manager

ICONBUTTON.interior:

Pattern number of the interior of the icon button (PATTERN1 to PATTERN20).

ICONBUTTON.pUp:

Pointer to the stroke list defining the deselected or up state of the icon button. (Refer to the form manager section of this manual for details on the stroke list)

ICONBUTTON.pDown:

Pointer to the stroke list defining the selected or down state of the icon button. (Refer to the form manager section of this manual for details on the stroke list)

```
/* Check box Component */
struct checkbox_defn
{
    CMP_HEADER    header;
    char          bState; /* checked/unchecked flag */
};
typedef struct checkbox_defn CHECKBOX;
```

CHECKBOX.header:

This header is a header structure which provides information to the component manager such as where the check box will be displayed. Refer to the CMP_HEADER documentation for a further discussion of component headers.

header.maprect:

Extent constants:

CB_XEXT 2*CHAR_XEXT - x extent of a check box.

CB_YEXT 1*CHAR_YEXT - y extent of a check box.

CHECKBOX.bState:

Indicates whether the check box is checked or unchecked (CB_CHECKED or CB_UNCHECKED).

```
CB_CHECKED      SELECTED
CB_UNCHECKED    DESELECTED
```

The list box structure is as follows:

```
/* List box Component */
struct listbox_defn
{
    CMP_HEADER    header;
    char          bTitle; /* title selectable flag */
    char          bScroll; /* scroll direction flag */
    char          bAlphabetize; /* list alphabetization flag */
    char          bMulti; /* multi select flag */
    char          border; /* border type */
    char          bgnd_color; /* background color */
    char          fgnd_color; /* foreground color */
    unsigned char top_string; /* number of top item in window */
    unsigned char selected; /* list box selected item */
    unsigned char *pString; /* list box title */
    unsigned char nItems; /* number of strings */
    char          *pSelected; /* pointer to selected list */
    unsigned char **pItems; /* pointer to string list */
};
typedef struct listbox_defn LISTBOX;
```

Listbox header:

DeskMate Technical Reference

Component Manager

Header is a structure that provides information to the component manager, such as where the list box is to be displayed. Refer to the CMP_HEADER documentation for a further discussion of component headers.

```
LB_BASE_XEXT      2*CHAR_XEXT
LB_BASE_YEXT      2*CHAR_YEXT
xext:
```

Use the following formula for defining the xext.

Vertical list box:

```
LB_BASE_XEXT + ( n * CHAR_XEXT )
```

Horizontal list box:

```
2 * (LB_BASE_XEXT + ( n * CHAR_XEXT ) )
```

Where 'n' is the number of characters in the longest string.

yext:

Use the following formula for defining the yext.

```
LB_BASE_YEXT + ( n * CHAR_YEXT )
```

Where 'n' is the number of lines to display in the window at any one time.

Listbox bTitle:

Indicates whether the cursor may be positioned on the title. This flag should be set to ENABLED or DISABLED. If bTitle is enabled, the bMulti flag must be disabled.

Listbox bScroll:

Defines whether the list box should scroll horizontally (two column format) or vertically (single column format). Set bScroll to LB_VERT_SCROLL or LB_HORZ_SCROLL.

```
LB_VERT_SCROLL    0      /* single column vertical scrolling */
LB_HORZ_SCROLL    1      /* 2 column horizontal scrolling */
```

Listbox bAlphabetize:

Indicates whether the list box items should be sorted before they are displayed. Set bAlphabetize to ENABLED or DISABLED.

Listbox bMulti:

Indicates whether the user can select multiple items. Set bMulti to ENABLED or DISABLED. If bMulti is enabled, the flags pointed to by pSelected reflect the items that are selected.

```
/* List box in multi-select mode mask for */
/* turning flashing underline on or off */
LB_IN_MULTII      0x02
```

Listbox border:

Enables a border to be draw around the list box. The border types allowed are those supported by **vid_draw_frame** (FLAT_BOX, RAISED_BOX, PYRAMID, DEST_FLAT_BOX, and LISTBOX_BOX. Set border to LB_NO_FRAME if no frame is desired. Set border to LB_NO_BORDER if neither a frame nor an inside border is desired.

```
LB_NO_FRAME       -1      /* no frame around list box */
LB_NO_BORDER      -2      /* no border or frame around list box */
```

Listbox bgnd_color

Specifies the background color for the list box. If bgnd_color and fgnd_color are the same, the current foreground and background colors are used.

```
LB_NO_COLOR       CSR_DEFAULT
```

Listbox fgnd_color

Specifies the foreground color for the list box. To use the default colors define this as LB_NO_COLOR.

DeskMate Technical Reference Component Manager

LB_NO_COLOR CSR_DEFAULT

Listbox top_string

Specifies the item to be displayed at the top of the list box. If the selected item is outside the list box window, top_string will be adjusted to make it visible.

/* y extent from top of list box to top of list */
LB_TITLE_YEXT CHAR_YEXT*3/2

Listbox selected

Specifies the item on which the cursor is positioned. If bMulti is disabled, selected is also the currently selected item. If bMulti is enabled, the application must check pSelected to determine which items are selected. More than one item in the list box might be selected.

Listbox pString

Points to the list box title. pString should be set to zero if a title is not desired.

Listbox nItems

Contains the number of items in the list box.

Listbox pSelected

Points to an array of one byte flags that show which items are selected. These flags should be set to SELECTED or DESELECTED. The array should contain one byte for each item in the list box. The title does not have an entry in the array because the title cannot be selected.

Listbox pItem

Points to an array of string pointers. These string pointers point to the item strings.

/* Edit field Component */

An edit field structure specifies the type of data that will be accepted in the edit field, defines the appearance and location of the edit field, and defines how much of the field is visible on the screen. The edit field structure is defined as follows:

```
struct editfield_defn
{
    CMP HEADER      header;
    char            type;            /* edit field type */
    char            bHighlight;      /* flag to highlight field on run */
    char            dec_places;      /* num places to the right of . */
    MAPRECT        edit_maprect;    /* visible part */
    char            attr;            /* char attribute to use */
    int            cursor_offset;    /* cursor offset into string */
    int            end_offset;       /* offset to end of string */
    int            select_offset;    /* offset to select start */
    int            select_length;    /* length of selected area */
    char            terminator;      /* terminating character of string */
    unsigned char   *pBuffer;        /* pointer to edit string */
    unsigned char   *pFormat;        /* pointer to format string */
    char            scroll_type;      /* direction of scroll */
    int            scroll_length;    /* number of chars to scroll */
    int            scroll_height;    /* number of rows to scroll */
    char            bNoEdit;        /* editable field flag */
    char            border;          /* border type for edit field */
};
typedef struct editfield_defn EDITFIELD;
```

Editfield header:

DeskMate Technical Reference Component Manager

Header is a structure which provides information about the edit field to the component manager. For example, it specifies the location at which the field will be displayed. Refer to the CMP_HEADER documentation for a further discussion of component headers.

Editfield type:

Indicates which type of edit field is to be run. The following types are available:

EF_STATIC	0x00	- static, single line edit field.
EF_EXPAND	0x01	- single line edit field that reverses the foreground and background colors in the field. Expands the size of the field on insert.
EF_NUMBER	0x04	- right justified single line edit field which will accept numeric data and a plus or a minus sign.
EF_EXTNUM	0x06	- right justified single line edit field which will accept only numeric data and a decimal point.
EF_WORD_WRAP	0x08	- multiple line edit field which does word wrapping.
EF_NO_WORD_WRAP	0x10	- multiple line edit field which does not word wrap.

Editfield bHighlight:

This element should be set to EF_HIGHLIGHT or EF_NO_HIGHLIGHT. When it is set to EF_HIGHLIGHT the entire edit field is displayed with the foreground and background colors reversed.

EF_HIGHLIGHT	ENABLED
EF_NO_HIGHLIGHT	DISABLED

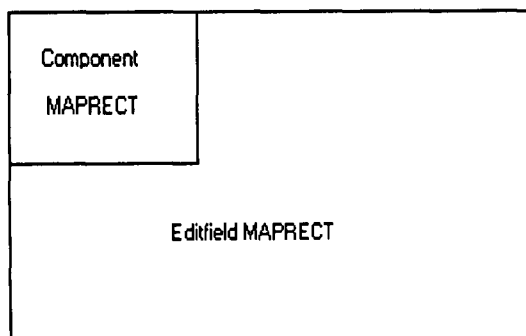
Editfield dec_places:

Specify a value for this element any time the edit field type is set to EF_NUMBER. Specify 0-9 to indicate the number of fixed decimal places in the field.

Editfield edit_maprect:

The maprect describes the size and origin of the edit field buffer. These values should be in world coordinates. The xorg and yorg parameters are the x and y origin of the buffer relative to edit field window. Initially, these values should be set to zero. They will become negative if text scrolls. The text and yext parameters describe the length and height of the edit field buffer. If xext and yext in this structure are equal to the xext and yext values in CMP_HEADER, no scrolling will occur. Scrolling is not allowed for EF_NUMBER, and EF_EXTNUM.

For example:



DeskMate Technical Reference Component Manager

Editfield attr:

Allows the edit field text to be displayed with the character attributes supported by **vid_set_char_attr**. The attributes supported are: NORMAL, BOLD, UNDERLINE, INVERSE, GRAYED, and TRANSPARENT.

Editfield cursor_offset:

Specifies the offset of the cursor location relative to the beginning of the text. For example, to place the cursor on the third character, set **cursor_offset** to 2. If **cursor_offset** is set to **EF_SELECT_ALL**, then all text in the buffer will be selected and the cursor will be set to position zero. **cursor_offset** **must not** exceed the number of text characters in the edit field buffer. Upon entry to a dialog box or an edit field, set **cursor_offset** to **EF_SELECT_ALL**.

EF_SELECT_ALL -1

Editfield end_offset:

Contains the number of text characters in the edit field buffer. This value does not need to be initialize, it is returned when an edit field is run or drawn.

Editfield select_offset:

Specifies the offset of the first selected character.

Editfield select_length:

Specifies the length of the selected text.

Editfield terminator:

Allows a terminating character to be displayed at the end of the text to show where the text ends. If the edit field's buffer is filled, the character will not be displayed. The edit field will display the terminating character only when the edit field type is set to **EF_STATIC** or **EF_EXPAND**. If a terminating character is not desired, set the terminator to zero.

Editfield pBuffer:

Pointer to the edit field text (buffer). The text must be null terminated. If no text is in the buffer, the first character in the buffer must be a null. A single-line edit field's buffer size should be one (for the null terminator) plus the number of characters that fit into the edit field. To calculate the size of a multiple line edit field's buffer use the following formula:

$((\text{number of characters in a line} + 2) * \text{number of lines}) + 1$

Editfield pFormat:

Allows the user to enter formatted data, such as a social security number or date. This option can only be used with the **EF_STATIC**, **EF_NUMBER**, and **EF_EXTNUM** edit field types. **pFormat** is a pointer to a null terminated string with the same number of characters as will fit into the edit field. The string should contain a 0x16 wherever there is no format character. Format characters can be valued from 0x20 to 0xff. If a format string is not desired, then **pFormat** should be set to zero. See example 2. A format string is required if **dec_places** is not zero. The format string must contain the decimal point at the appropriate location.

Editfield scroll_type:

This should be initialized to **EF_NO_SCROLL** or **EF_SCROLL**. Scroll type should be set to **EF_NO_SCROLL** if the edit field does not scroll or if it is desired to have the edit field scroll automatically. **scroll_type** should be initialized to **EF_SCROLL** if the application desires to handle scrolling. If **EF_SCROLL** is set, when scrolling is required the edit field will return with a **CMP_SCROLLED** status and **scroll_type** will be set to reflect the direction of the scroll as listed

DeskMate Technical Reference Component Manager

below. If the edit field returns with `CMP_SCROLLLED`, `scroll_type` must be reset to `EF_SCROLL`, before rerunning the edit field.

<code>EF_SCROLL_UP</code>	0
<code>EF_SCROLL_DOWN</code>	1
<code>EF_SCROLL_LEFT</code>	2
<code>EF_SCROLL_RIGHT</code>	3
<code>EF_NO_SCROLL</code>	4
<code>EF_SCROLL</code>	5

Edit fields cannot support both horizontally and vertically scrolling edit fields simultaneously.

Editfield `scroll_length`:

Aids in recalculating the edit maprect's xorg when the edit field returns with a `CMP_SCROLLLED` status.

Editfield `scroll_height`:

Aids in recalculating the edit maprect's yorg when the edit field returns with a `CMP_SCROLLLED` status.

Editfield `bNoEdit`:

This option should be set to `EF_EDITABLE` or `EF_NO_EDIT` to indicate whether the user can modify the text in the edit field. If `EF_NO_EDIT` is specified, the user will only be able to move the cursor and select and copy text. Keystrokes for adding or deleting text will be ignored.

<code>EF_EDITABLE</code>	<code>DESELECTED</code>
<code>EF_NO_EDIT</code>	<code>SELECTED</code>

Editfield `border`:

Allows a border to be draw around the edit field. The border types allowed are those supported by `vid_draw_frame`: `FLAT_BOX`, `RAISED_BOX`, `PYRAMID`, `DEST_FLAT_BOX`, and `LISTBOX_BOX`. If a frame is not desired, set border to `EF_NO_FRAME`.

<code>EF_NO_FRAME</code>	<code>CSR_ERROR</code>
--------------------------	------------------------

`/* Menu bar Component */`

Three types of structures are important when creating menus: the structure for the menu bar, a structure for each menu in the menu bar, and a structure for each item in the menu.

`struct menuitem_defn`

```
{
    char          type;           /* item type */
    char          bEnabled;       /* enable (grayed) flag for item */
    char          bChecked;       /* check flag */
    unsigned int  accel;          /* accelerator for item */
    unsigned int  return_code;    /* return code for item */
    char          group;          /* group number of item */
    char          pattern;        /* pattern to use as the item */
    unsigned char *pString;       /* pointer to item name string */
};
```

`typedef struct menuitem_defn MENUITEM;`

`MENUITEM.type`:

Defines the type of menu item.

<code>MB_STRING</code>	0	<code>/* item is a string */</code>
<code>MB_PATTERN</code>	1	<code>/* item is a pattern */</code>

`MENUITEM.bEnabled`:

DeskMate Technical Reference Component Manager

Enable flag for the menu item (ENABLED or DISABLED).

MENUITEM.bChecked:

Indicates whether or not the menu item is checked or unchecked.

DeskMate Technical Reference Component Manager

MB_CHECKED SELECTED
MB_UNCHECKED DESELECTED

MENUITEM.accel:

Keyboard accelerator for this menu item. MB_ACCEL is not an accelerator that can be equated to a menu item accelerator. It is used internally to pull down the F10 menu.

NO_ACCEL -1 /* no accelerator */
MB_ACCEL ALT_SPACE_KEY /* pulls down the F10 menu */

MENUITEM.return_code:

Return code for the menu item.

MENUITEM.group:

Group number of the menu item. The group numbers **MUST** be sequential and **MUST** begin with MB_GROUP1 for each menu. Changing the group number automatically draws a separator line in the menu item list.

MB_GROUP1 0
MB_GROUP2 1
MB_GROUP3 2
MB_GROUP4 3
MB_GROUP5 4
MB_GROUP6 5
MB_GROUP7 6
MB_GROUP8 7
MB_GROUP9 8

MENUITEM.pattern:

Pattern number of the menu item (PATTERN1 to PATTERN20). Pattern is only valid if the type element is MB_PATTERN.

MENUITEM.pString:

Pointer to the menu item string. pString is valid only if the type element is MB_STRING.

```
struct menu_defn
{
    int          xext;          /* x extent of menu */
    unsigned char *pString;     /* pointer to button name string */
    char         nItems;        /* number of items in the menu */
    MENUITEM     *pItems;       /* pointer to MENU ITEM structures */
};
typedef struct menu_defn MENU;
```

MENU.xext:

World unit x extent of the menu. Use the formula described below to define this element.

MENU.pString:

Pointer to menu button label string.

MENU.nItems:

Number of items in the menu.

MENU.pItems:

Pointer to the MENUITEM structures for the menu.

```
struct menubar_defn
{
```

DeskMate Technical Reference Component Manager

```

CMP HEADER  header;
char        bFlags;           /* arrow and top line flags */
char        bRedraw;          /* visible redraw flag */
char        nMenus;           /* number of menus on the menu bar */
MENU        *pMenus;          /* pointer to MENU structures */
char        pattern;          /* background pattern of the menubar */
char        bStatus;          /* process help only flag */
};
typedef struct menubar_defn MENUBAR;

```

MENUBAR.header:

This header is a header structure that provides information to the component manager. Refer to the CMP HEADER documentation for a further discussion of component headers.

```

MB_XORG      0                /* default x origin of a menu bar */
MB_YORG      290              /* default y origin of a menu bar */
MB_XEXT      80*CHAR_XEXT     /* default x extent of a menu bar */
MB_YEXT      345              /* y extent of menu bars */

```

```

/* Menu width */
MB_BASE_XEXT 6*CHAR_XEXT     /* base x extent of a menu */

```

MENUBAR.bFlags:

Indicates which buttons are to be available on the menu bar, as defined below. bFlags shows what kind of icons appear at the edge of the menu bar. Icons are defined in CSRCMPS.H. For example: MB_TANDY + MB_ALL_ARROWS means the Tandy Icon (Accessories Menu) and all four of the arrow icons will show on the menu bar line. It is strongly suggested that when running a menu bar as a component within a dialog box, the message menu and the accessory menu not be displayed.

```

/* Flag masks */
MB_PLAIN      0                /* no Tandy, help or arrow buttons */
MB_HELP_BUTTON 1                /* help button */
MB_RIGHT_ARROW 2                /* right arrow button */
MB_LEFT_ARROW 4                /* left arrow button */
MB_DOWN_ARROW 8                /* down arrow button */
MB_UP_ARROW   16               /* up arrow button */
MB_TANDY      32                /* Accessories icon button */
MB_ALARM      64                /* alarm button */
MB_VERT_ARROWS MB_UP_ARROW+MB_DOWN_ARROW
MB_HORZ_ARROWS MB_LEFT_ARROW+MB_RIGHT_ARROW
MB_ALL_ARROWS MB_VERT_ARROWS+MB_HORZ_ARROWS

```

MENUBAR.bRedraw:

Indicates whether to redraw the menu buttons on cmp_draw or mb_draw (SELECTED or DESELECTED). This element must be SELECTED the first time mb_draw is used, but may be deselected on subsequent calls. Also, if the application only wishes to change the enable or checked flags of the menu items, cmp_open will select this element and cmp_draw will deselect it.

```

MB_REDRAW      SELECTED        /* redraw menu bar */
MB_NO_REDRAW    DESELECTED     /* only redefine enabled/disabled items */

```

MENUBAR.nMenus:

Number of labeled menu button on the menu bar.

MENUBAR.pMenus:

Pointer to the MENU structures for the menu bar.

MENUBAR.pattern:

DeskMate Technical Reference Component Manager

Background pattern number of the menu bar (PATTERN1 to PATTERN20).

MENUBAR.bStatus:

Alarm menu access mask for bStatus element of the MENUBAR structure.

/* Alarm menu access mask for bStatus element of MENU BAR */

/* structure, it is used internally by the alarm manager */

MB_ALARM_ACCESS 0x01

/* World coordinate of the last default main menu bar pixel */

MB_BOTTOM_YORG MB_YORG+MB_YEXT-1

/* mb_add_alarm return codes */

MB_NOT_NOTIFIED 0x01 /* alarm added but User not notified */

MB_ALARM_EXISTS 0x02 /* alarm exists, not added again */

/*-----*/

/* SCROLL BAR COMPONENT */

/*-----*/

struct scroll_bar_defn

```
{
    CMP_HEADER header;
    char bType; /* type of scroll bar */
    int nUnits; /* total units in scroll bar */
    int elevator_size; /* size of elevator in world */
    /* coordinates. */
    int arrow_size; /* size of arrow icons in world */
    /* coordinates. */
    int page_inc; /* units for page up/down */
    int arrow_inc; /* units to increment for arrow */
    int position; /* position of elevator in units */
    int pos_change; /* position changed in units */
    char fgnd_color; /* foreground color of scroll bar */
    char bgnd_color; /* background color of scroll bar */
    char efgnd_color; /* elevator fore-ground color */
    char ebgnd_color; /* elevator back-ground color */
    int (far *pHomeImage)(); /* pointer to IMAGE structure of */
    /* upper left box */
    int (far *pEndImage)(); /* pointer to IMAGE structure of */
    /* lower right box */
};
```

typedef struct scroll_bar_defn SCROLL_BAR;

CMP_HEADER.header:

The header is a component header structure which provides information to the component manager such as where the scroll bar will be displayed. Refer to the CMP_HEADER documentation for a further discussion of component headers.

CMP_HEADER.type:

Specifies that this component is the scroll bar. The value returned from add_scroll_bar_cmp should be stored here before the scroll bar can run, opened, closed, or drawn.

CMP_HEADER.bEnabled:

Specifies if the scroll bar should be disabled. If a disabled scroll bar is drawn, it will be displayed with the grayed attribute. A disabled component cannot be run.

CMP_HEADER.xorg:

Specifies the X origin of where the scroll bar will be displayed.

DeskMate Technical Reference Component Manager

CMP_HEADER.yorg:
Specifies the Y origin of where the scroll bar will be displayed.

CMP_HEADER.xext:
Specifies the width of a vertical scroll bar or the length of a horizontal scroll bar. If a vertical scroll bar is to be used, SBAR_VERTICAL should be specified.

CMP_HEADER.yext:
Specifies the length of a vertical scroll bar or the height of a horizontal scroll bar. If a horizontal scroll bar is used, SBAR_HORIZONTAL should be specified.

CMP_HEADER.accel:
Specifies the scroll bar's accelerator.

CMP_HEADER.return_code:
Specifies the scroll bar's return code.

SCROLL_BAR.bType:
Specifies the desired type of scroll bar. These values should be or'ed together:

SBAR_VERTICAL	0	bit 0 cleared vertical scroll bar
SBAR_HORIZONTAL	1	bit 0 set horizontal scroll bar
SBAR_NO_ARROWS	4	mask for no home and end icon box
SBAR_USER_DEFINED	8	mask for user defined icon box

CMP_HEADER.nUnits:
Total number of units in scroll bar.

CMP_HEADER.elevator_size:
Size of the elevator in world coordinates. bType should be set to use SBAR_ELEVATOR_FIXED.

CMP_HEADER.arrow_size:
Size of the arrow icons in world coordinates. This can be undefined if SBAR_NO_ARROWS is used. If a vertical scroll bar is used, arrow_size should be defined as SCRL_DEFAULT_VERT_ARROW. If a horizontal scroll bar is used, arrow_size should be set to SCRL_DEFAULT_HORZ_ARROW.

CMP_HEADER.page_inc:
Number of units the elevator travels when a mouse click occurs on the scroll bar above or below the elevator. This is analogous to scrolling up and down a page at a time, or right and left for a horizontal scroll bar.

CMP_HEADER.arrow_inc:
Number of units the elevator travels for a single arrow event. This is typically used to travel up and down one line at a time, or right and left for horizontal scroll bars.

CMP_HEADER.position:
Position of the elevator. This is in units.

CMP_HEADER.pos_change:
The number of units the elevator has traveled while running the scroll bar is returned here. If a negative value is stored here upon returning from cmp_run, the elevator traveled upwards. If pos_change is positive, the elevator traveled downwards.

DeskMate Technical Reference Component Manager

CMP_HEADER.fgnd_color:
Specifies the foreground color for the scroll bar.

CMP_HEADER.bgnd_color:
Specifies the background color for the scroll bar.

CMP_HEADER.efgnd_color:
Specifies the foreground color for the scroll bar's elevator and arrow icons.

CMP_HEADER.ebgnd_color:
Specifies the background color for the scroll bar's elevator and arrow icons.

Actually, the arrows use fgnd_color and bgnd_color.

CMP_HEADER.(far *pHomeImage)():
Far pointer to an IMAGE structure for the top/left arrow icon. This is only used when type is set to SBAR_USER_DEFINED.

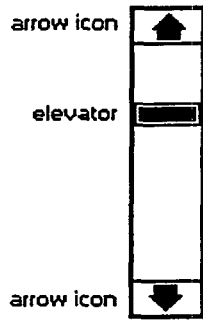
CMP_HEADER.(far *pEndImage)():
Far pointer to an IMAGE structure for the bottom/right arrow icon. This is only used when type is set to SBAR_USER_DEFINED.

SCRL_DEFAULT_VERT_WIDTH	200
SCRL_DEFAULT_VERT_ARROW	220
SCRL_DEFAULT_HORZ_HEIGHT	220
SCRL_DEFAULT_HORZ_ARROW	200
SCRL_DEFAULT_ELEVATOR_SIZE	210

Example vertical scroll bar:

```
SCROLL_BAR test_scroll_bar = {
    0,                                /* set after adding the cmp */
    ENABLED,                          /* component will be available */
    0, 0,                            /* origin is 0,0 */
    SCRL_DEFAULT_VERT_WIDTH,          /* use standard width */
    CHAR_XEXT * 12,                  /* scroll bar is 12 lines high */
    SCROLL_BAR_ACCEL,                 /* the accel & ret code should */
    SCROLL_BAR_RET_CODE,              /* be defined by the app */
    SBAR_VERTICAL,                   /* vertical scroll bar */
    100,                             /* coverage is 100 units total */
    SCRL_DEFAULT_ELEVATOR_SIZE,        /* use standard size */
    SCRL_DEFAULT_VERT_ARROW,          /* use standard size */
    10,                              /* page increment is 10 units */
    1,                               /* arrow increment */
    0,                               /* start elevator at top */
    0,                               /* pos_change is returned */
    COLOR1,COLOR2,                   /* scroll bar colors */
    COLOR3,COLOR4,                   /* elevator & arrow colors */
    0, 0                             /* SBAR_USER_DEFINED not given */
};
```

DeskMate Technical Reference Component Manager



DeskMate Technical Reference Component Manager

Application Defined General Description/Notes

Application defined components are described with a structure that includes information such as the components location and size, and the command code to return when the user selects the component. The ability to design components gives the application the flexibility to use whatever method of user interface best suits the immediate task.

DeskMate Technical Reference Component Manager

Application Defined Structures/Defines

Application defined component structures are defined by the application that uses them. The only restriction is that the user defined component structure **MUST** begin with a CMP_HEADER structure.

```
/* Component Manager */
struct cmp_routines_defn
{
    int    (far *pOpen)();      /* pointer to open routine for the
                                component */
    int    (far *pDraw)();     /* pointer to draw routine for the
                                component */
    int    (far *pRun)();      /* pointer to run routine for the
                                component */
    int    (far *pClose)();    /* pointer to close routine for
                                component */
    int    (far *pHelp)();     /* pointer to routine to return help
                                string */
};
typedef struct cmp_routines_defn CMP_ROUTINES;
```

pOpen:

The far pointer to the open routine in the application for an application defined component.

pDraw:

The far pointer to the draw routine in the application for an application defined component.

pRun:

The far pointer to the run routine for an application defined component.

pClose:

The far pointer to the close routine for an application defined component.

pHelp:

The far pointer to the routine that returns the help string.

```
/* Null routine */
CMP_NO_ROUTINE    0      /* indicates that there is no further */
                        /* processing for the component beyond the */
                        /* Component Manager processing for that */
                        /* routine. */
```

For example:

```
int far Cmp_Open_Editfield();
int far Cmp_Draw_Editfield();
int far Cmp_Run_Editfield();
int far Cmp_Close_Editfield();

CMP_ROUTINES EditfieldRoutines =
{
    Cmp_Open_Editfield,
    Cmp_Draw_Editfield,
    Cmp_Run_Editfield,
    Cmp_Close_Editfield,
    CMP_NO_ROUTINE
};
```

DeskMate Technical Reference

Component Manager

Application Defined

cmp_add (pCmpRoutines)
CMP_ROUTINES *pCmpRoutines;

cmp_add registers an application defined component with the component manager.

Input Parameters

pCmpRoutines is a pointer to a CMP_ROUTINES structure, which contain the pointers to the application's component functions.

Return Value

Component type which will be used by the application to open, draw, run and close the components. The CMP_HEADER structure the type element's value.

CSR_ERROR if the component could not be added to the list. The maximum number of components that can be added is eight.

Special Notes

cmp_add will copy the passed structure into the component manager's component table. Any 0L's (zero long) will be converted to a long pointer to RETF within the CSR code segment. The component table is limited to 8 application defined entries.

Assembly language programmers should note that the added component routines must preserve the ES, DX, and AX segment registers. These registers contain the component handle and the window handle in your user defined routine.

The user must use the component type returned by **cmp_add**, in CMP_HEADER when calling **cmp_open**, **cmp_close**, **cmp_draw**, or **cmp_run**.

Example

```
/* Special Editfield definition */
struct my_editfield_defn
{
    EDITFIELD    Editfield;
    int          hEditfield;
};
typedef struct my_editfield_defn MY_EDITFIELD;

char filename_buff[MAX_FILE_NAME_SIZE+1];

MY_EDITFIELD NewFileEF =
{
    CMP_EDIT,
    ENABLED,
    12*CHAR_XEXT,
    1*CHAR_YEXT,
    20*CHAR_XEXT,
    1*CHAR_YEXT,
    0,
    /* header.type */
    /* header.bEnabled */
    /* header.maprect.xorg */
    /* header.maprect.yorg */
    /* header.xext */
    /* header.yext */
    /* header.accel */
}
```

DeskMate Technical Reference Component Manager

```

EF tag,                                /* header.return_code */
EF_STATIC,                             /* type */
EF_NO_HIGHLIGHT,                       /* bHighlight */
0, 0,                                  /* dec places (places to right of decimal */
0, 0,                                  /* MAPRECT.xorg, .yorg (editable area) */
(MAX FILE_NAME_SIZE+1)*CHAR_XEXT, /* MAPRECT.xext */
1*CHAR_YEXT,                           /* MAPRECT.yext */
NORMAL,                                /* attr (char attribute) */
0, 0,                                  /* cursor_offset, end_offset */
0, 0,                                  /* select_offset, select_length */
CSR_NULL,                              /* terminator */
filename buff,                         /* *pBuffer (pointer to EF buffer) */
(char *)0,                             /* *pFormat */
EF_NO_SCROLL,                          /* scroll_type */
0, 0,                                  /* scroll_length, scroll_height */
EF_EDITABLE,                           /* bNoEdit */
RAISED_BOX,                           /* border */
0                                       /* hEditfield */
};

main(argc, argv)
int    argc;
char   *argv[];
{
    /* Bind to the Core Services Resource */
    if ( csr_init() == CSR_ERROR )
        /* failure to bind to the CSR, could not find/load resource */
        exit(1);

    /* Get the component type for this special editfield type */
    NewFileEF.Editfield.header.type = cmp_add(&EditfieldRoutines);

    /* Rest of program... */
    /* Run dialog box which has this type of editfield defined in it */
    /* The application component routines will be called */
}

```

DeskMate Technical Reference
Component Manager
Application Defined

cmp_close (Handle, pCmp)

int Handle;

APPL_DEFINED *pCmp; < - *actually void *pCmp*

cmp_close terminates an application defined component.

Input Parameters

Handle is the handle of the component to close, as returned by **cmp_open**.

pCmp is a pointer to the application defined component structure, which **MUST** begin with a **CMP_HEADER** structure.

Return Value

<int>

CSR_ERROR if the **Handle** is invalid.

Special Notes

cmp_close closes the hot spot for an application defined component. It creates the component's working window and calls the application supplied close routine for the component. After returning from the component close routine, **cmp_close** closes the working window of the component. **CMP_NULL_HANDLE** can **NOT** be substituted for the handle in **cmp_close**.

The application should supply a close function if any special processing is required when the application defined component is closed.

Example

The application defined component is passed two parameters for example:

```
int far Cmp_Close_Editfield(hEditfield, pEditfield)
int hEditfield;
MY_EDITFIELD *pEditfield;
{
    int          my_type;

    my_type = pEditfield->Editfield.header.type;

    pEditfield->Editfield.header.type = CMP_EDIT;
    cmp_close( pEditfield->hEditfield, pEditfield );
    pEditfield->Editfield.header.type = my_type;
}
```

hEditfield is the handle to the component's working window.

pEditfield is the pointer to the application defined component structure.

DeskMate Technical Reference
Component Manager
Application Defined

cmp_delete (CmpType)
int CmpType;

cmp_delete removes a registered application defined component from the component manager.

Input Parameters

CmpType is the type for the component to delete as defined by the type returned by **cmp_add** for the desired component.

Return Value

CSR_ERROR if **CmpType** was incorrect.

Special Notes

The application cannot delete CSR defined (predefined) components. The predefined CSR component's are CMP_CHECKBOX, CMP_EDIT, CMP_ICONBUTTON, CMP_LISTBOX, CMP_MENUBAR, CMP_PUSHBUTTON, and CMP_RADIOBUTTONS.

DeskMate Technical Reference
Component Manager
Application Defined

cmp_draw (handle, pCmp)

int handle;

APPL_DEFINED *pCmp; ← actually void *pCmp

cmp_draw draws an application defined component.

Input Parameters

Handle is the handle of the component to draw, as returned by **cmp_open**.

pCmp is a pointer to the application defined component structure, which **MUST** begin with a **CMP_HEADER** structure.

Return Value

CSR_ERROR if the component could not be drawn.

Special Notes

To draw a component which has not been opened, use **CMP_NULL_HANDLE** as the handle. If the handle is **CMP_NULL_HANDLE** then no hot spot processing is done. **cmp_draw** enables or disables the hot spot of the specified component, opens its working window and calls the draw routine of the component. After returning from the application supplied draw routine of the component, **cmp_draw** closes the component's working window.

The application must supply a routine which either draws the component or calls other **cmp_draw** routines to draw the application defined component.

Example

The application defined component routine is passed two parameters for example:

```
int far Cmp Draw_Editfield(hEditfield, pEditfield)
int hEditfield;
MY_EDITFIELD *pEditfield;
{
    int          my_type;

    my_type = pEditfield->Editfield.header.type;

    pEditfield->Editfield.header.type = CMP_EDIT;
    win_activate( hEditfield );
    cmp_draw( pEditfield->hEditfield, pEditfield );
    pEditfield->Editfield.header.type = my_type;
}
```

hEditfield is the handle to the component's working window.
pEditfield is the pointer to the application defined component structure.

← The working window does not seem to be the component's window - use **win_get_active()** instead.

DeskMate Technical Reference

Component Manager

Application Defined

cmp_open (pCmp)

APPL_DEFINED *pCmp; ← actually void *pCmp

cmp_open initializes an application defined component.

Input Parameters

pCmp is a pointer to the application defined component structure, which **MUST** begin with a **CMP_HEADER** structure.

Return Value

The handle of the component.

CSR_ERROR if the component could not be opened.

Special Notes

cmp_open does not draw the component.

cmp_open creates a hot spot and window (as defined in the component header) and calls the application's open routine for the component. After returning from the open routine of the component, **cmp_open** closes the component's working window.

Example

The application defined component is passed three parameters for example:

```
int far Cmp_Open_Editfield(hParent, pEditfield, hChild)
int      hParent;
MY EDITFIELD *pEditfield;
int      hChild;
{
    int      my_type;

    my_type = pEditfield->Editfield.header.type;

    pEditfield->Editfield.header.type = CMP_EDIT;
    pEditfield->hEditfield = cmp_open( pEditfield );
    pEditfield->Editfield.header.type = my_type;
}
```

hParent is the handle to the parent window.

pEditfield is the pointer to the application defined component structure.

hChild is the handle to the component's working window.

DeskMate Technical Reference

Component Manager

Application Defined

cmp_run (Handle, pCmp)

int Handle;

APPL_DEFINED *pCmp; *← actually void *pCmp*

cmp_run runs an application defined component.

Input Parameters

Handle is the handle of the component to close, as returned by **cmp_open**.

pCmp is a pointer to the user defined component structure, which **MUST** begin with a CMP_HEADER structure.

Return Value

CMP_NO_ACTION if no action was taken on the component. Exit was forced by the TAB key, the BACKTAB key or an unused arrow key.

CMP_ACTION if the status of the component has changed.

CMP_SCROLLED if scroll has occurred. This return code is generated by the edit field component only.

CMP_CANCEL if no action was taken on the component. Exit was forced by ESC key.

CMP_DISABLED if the component was disabled and was not processed.

CMP_GO if the user invoked go. In a dialog box, the application should invoke the most affirmative action. Return key or double click with mouse.

CMP_ACTION_IN_EVENT if the component return code is in the event manager event queue. This return code is generated by the menu bar component only.

CMP_SELECT_CHANGE when the component selection has changed. For example:

1. An edit field has started or ended selection
2. A radio button has been chosen
3. A new list box item has been selected or de-selected

CSR_ERROR if the component could not be run.

CMP_APPL if the user executes a task switch or runs an accessory.

Special Notes

An unopened component may be processed by passing CMP_NULL_HANDLE as handle.

cmp_run returns CMP_DISABLED if the component is DISABLED, otherwise it creates the components working window and calls the components run routine. After returning from the run routine **cmp_run** closes the working window.

Example

The application defined component is passed two parameters for example:

```
int far Cmp Run Editfield(hEditfield, pEditfield)
int hEditfield;
MY_EDITFIELD *pEditfield;
```

DeskMate Technical Reference Component Manager

```
{  
    int          my_type, return_code;  
    my_type = pEditfield->Editfield.header.type;  
    pEditfield->Editfield.header.type = CMP_EDIT;  
    win_activate( hEditfield );  
    return_code = cmp_run( pEditfield->hEditfield, pEditfield );  
    pEditfield->Editfield.header.type = my_type;  
    /* return to the calling routine */  
    return(return_code);  
}
```

hEditfield is the handle to the component's working window.

pEditfield is the pointer to the application defined component structure.

DeskMate Technical Reference Component Manager

Pre-Defined Components General Description/Notes

Check box

Check boxes allow the user to set a function on or off. Any combination of check boxes can be checked at a time.

The state of a check box is either checked or not checked. An X appears inside a check box when the box is checked. When the box is not checked, no character or string appears inside.

Each check box should be followed by a static string that describes the option. The string should not end with a colon and phrases start with a capital letter.

Edit field

The edit field is a component that makes use of windows and hotspots to support text editing in any area of the screen. Edit fields are used primarily by dialog boxes for input fields, but are also available to applications for use in the work area.

An edit field can accept one or more lines of data. Single line edit fields can be used in dialog boxes or in the work area. Multiple line edit fields are not supported in dialog boxes.

Multiple line edit fields, which automatically word wrap, can contain up to 18K of data. Single line edit fields are limited to 255 characters and can contain static format characters that format data as the user enters it.

Several options enable custom edit fields. For example edit fields can be created that only accept a certain type of data, such as numeric data, or an edit field that only accepts data in a specific format, such as a phone number. The values assigned to elements of the edit field structure (described in this section) determine the type of data the edit field will accept.

Functions that open, draw, run, and close the edit field are described in this section. The routines used to access the clipboard and format the edit field buffer are described at the end of this section.

Any cursor motion within an edit field will deselect the text, if selected, and move the cursor.

Edit fields should be preceded by an identifying static string that ends with a colon. Edit fields in dialog boxes should always have a raised border.

If the application does not have the standard Cut/Copy/Paste accelerators defined, the editfield will automatically handle those operations. However, the editfield will not display error messages, such as not enough space in the field for the pasted data.

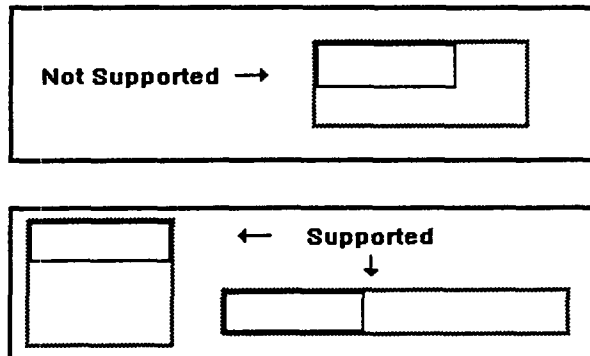
Scrolling within editfields

Scrolling edit fields need to reset the editing maprect origin back to zero when the beginning of the text is displayed. For example:

```
edit_maprect.xorg = 0;  
edit_maprect.yorg = 0;
```

DeskMate Technical Reference Component Manager

It is suggested that edit fields do not support both horizontal and vertical scrolling simultaneously. For example:



All edit fields should reset the following elements:

```
cursor_offset = EF_SELECT_ALL;  
select_offset = 0;  
select_length = 0;
```

For more information see the three edit field examples at the end of this General Description/Notes section.

Icon button

Icon buttons function like push buttons but their appearance is defined by a stroke list. An icon button component is a stroke list image bordered by a two or three dimensional rectangular frame. An icon button may be in one of two states, selected (pressed) or deselected (raised). In the selected state, the icon button will appear flat and can optionally be inverted. In the deselected state, it will be optionally flat or raised.

List box

A list box enables the user to select a string or set of strings from a list of choices. The list is displayed within a rectangular area defined by the component header in the list box structure.

A list box can scroll vertically or horizontally and can contain up to 255 items. Horizontal scrolling automatically uses a two column format and vertical scrolling uses a single column format.

If the alphabetize flag in the list box structure is **ENABLED**, the items will be alphabetized before the list box is drawn. Once alphabetized, the original order of the items is lost and `top_string` will be set to zero. If it is desired to alphabetize the list without drawing the list box, refer to `lb_sort_list` at the end of this section.

A list box is the only component that allows the user to "select and go," (select an item and immediately terminate the dialog box). When the user presses the enter key while in a list box, the current item is selected and the OK button is pushed. With a mouse, double clicking selects the current item and terminates the dialog box.

Menu Bar

This section discusses menu bars as a general user interface component. The DeskMate Menu Bar is a special menu bar. It is discussed in the "Menu Bar Manager" section of this manual.

DeskMate Technical Reference Component Manager

A menu bar is a group of menu buttons. Each button activates a single menu, a list of items or commands from which the user chooses. Every menu has a menu name which states the purpose as clearly as possible. DeskMate automatically assigns each menu a function key accelerator. When running the menu bar as a component do not include the F9 message menu, or the F10 accessory menu.

Push Button

Push buttons enable the user to control a dialog box or continue a process. Typically two push buttons are used in a dialog box. One, OK, enables the user to accept the dialog box input and continue with the function. The other, CANCEL, enables the user to cancel the dialog; no further action is taken with CANCEL.

Push button labels should be in all capital letters. Push buttons in a dialog box should all be the same size, the length of the longest label plus two spaces.

The OK, or affirmative action, button's accelerator is the enter key. The application should use the `DLG_OK_KEY` accelerator definition for this button. The CANCEL, or abort action, button's accelerator is the escape key. The application should use the `DLG_CANCEL_KEY` accelerator definition for this button.

Other push button accelerators may be defined by the application or defined automatically by the CSR. When the application specified `PB_DEFAULT_ACCEL`, as the accelerator for push buttons, the CSR will assign the accelerator for the push button. If the application wishes to use a different accelerator, it should be specified in the initial setup of the push buttons. All of these accelerators (mnemonics) are accessed via the ALT + <first letter> key sequence.

A flat push button will have a hot spot that is slightly taller than the flat push button image. The hot spot will actually be the size of a raised push button image.

Radio Buttons

Radio buttons allow the user to select a single option from a set of options. Radio buttons appear in groups, and the user can select only one button at a time. Radio button groups should be surrounded by static boxes to denote their grouping. Radio button group labels (static strings) should *not* end with a colon but should start with a capital letter. The cursor should always be displayed on the "pressed" button when the user tabs or arrows into the radio button group. Radio buttons are a mutually exclusive group (only one of the buttons may be pressed at a time).

When positioning individual radio buttons within a radio button group in a row/column format, the columns must be completed from left to right in order for arrow movements within the group to work properly. For example, if the application defines a radio button group of seven radio buttons in a 3 by 3 row/column format, the radio buttons must be arranged as follows:

```
x      x      x
x      x
x      x
```

These arrangements will not process arrow movements properly:

```
x      x      x      x      x      x      x      x      x
x      x      x      x      x      x      x      x      x
x      x      x      x      x      x      x      x      x
```


DeskMate Technical Reference Component Manager

The point here is that radio buttons must be arranged in such a way that any possible combination of enabled and disabled radio buttons will contain a direct vertical or horizontal path to each enabled radio button. If an enabled radio button does not have another enabled radio button directly above or below or to the right or left and it does not have focus, the user will not be able to move focus to it using the arrow keys.

Scroll Bar

The scroll bar provides the ability to conveniently page through text or other information using the mouse. The position of the elevator in the scroll bar reflects the position of the cursor in the edit field or whatever information being inspected. By moving the elevator within the scroll bar, the information on the screen will scroll to match the elevator's position. The elevator may be moved in three ways. First, a click on the top or bottom arrow icon will cause the elevator to move, in the specified direction, a line at a time. Secondly, by clicking the mouse on the scroll bar above or below the elevator (for a vertical bar), but not on the arrow icons, will cause the elevator to move a page at a time. Thirdly, the elevator may be positioned by placing the mouse pointer on the elevator and holding the mouse button down while positioning the elevator to desired place on the scroll bar. When the mouse button is released, the viewed information will scroll to match the position of the scroll bar's elevator.

DeskMate Technical Reference Component Manager

Edit field Examples:

Example 1:

This example creates a single-line edit field that is centered on the screen. The edit field is 40 characters across and does not scroll because the buffer length and the length of the window are the same.

```
unsigned char sample1_text[41] = {"sample one's text"};
EDITFIELD sample1_editfield[] =
{
    CMP_EDIT,                /* component type */
    ENABLED,                 /* bEnabled */
    19 * CHAR_XEXT,          /* xorg */
    12 * CHAR_YEXT,          /* yorg */
    40 * CHAR_XEXT,          /* text (window is 40 chars across) */
    CHAR_YEXT,              /* yext (window is 1 char high) */
    APP_TAG + CTRL_A,        /* accelerator */
    APP_TAG + CTRL_A,        /* return code */
    EF_STATIC,               /* edit type */
    EF_NO_HIGHLIGHT,         /* bHighlight is disabled */
    0,                       /* no dec_places, for # fields only */
    0, 0,                   /* origin of buffer */
    40 * CHAR_XEXT,          /* 40 characters in buffer */
    CHAR_YEXT,              /* single-line edit field */
    NORMAL,                  /* character attribute */
    0,                       /* position cursor at first char */
    0,                       /* end offset */
    0,                       /* select offset */
    0,                       /* select_length (nothing selected) */
    0,                       /* no terminating character */
    sample1_text,            /* pBuffer */
    0,                       /* no format string */
    EF_NO_SCROLL,            /* does not need scrolling */
    0,                       /* scroll height */
    0,                       /* scroll length */
    EF_EDITABLE,             /* text can be modified */
    EF_NO_FRAME              /* no border */
};
```

DeskMate Technical Reference Component Manager

Example 2:

This example creates and draws an edit field which is formatted for numeric input. The structure is set up to draw the edit field in the top left corner like:

\$,195.99

```

unsigned char sample2_text[11] = {"19599"};
unsigned char sample2_format_str[] = { '$', 0x16, 0x16, ',', 0x16,
                                         0x16, 0x16, 0x16, 0x16, 0 } /* 10 chars + null */

EDITFIELD sample2_editfield[] =
{
    CMP_EDIT,                /* component type */
    ENABLED,                 /* bEnabled */
    0,                       /* xorg */
    0,                       /* yorg */
    10 * CHAR_XEXT,          /* xext (window is 10 chars across) */
    CHAR_YEXT,               /* yext (window is 1 char high) */
    APP_TAG + CTRL_B,        /* accelerator */
    APP_TAG + CTRL_B,        /* return code */
    EF_NUMBER,               /* edit type */
    EF_NO_HIGHLIGHT,         /* bHighlight is disabled */
    2, 0,                    /* allow 2 places for cents */
    0, 0,                    /* origin of buffer */
    10 * CHAR_XEXT,          /* 10 characters in buffer */
    CHAR_YEXT,               /* single-line edit field (1 line high) */
    NORMAL,                  /* character attribute */
    0,                       /* position cursor at first character */
    0,                       /* end offset */
    0,                       /* select offset */
    0,                       /* select length (nothing selected) */
    0,                       /* no terminating character */
    sample2_text,            /* pBuffer */
    sample2_format_str,      /* format string 10 chars long */
    EF_NO_SCROLL,            /* does not need scrolling */
    0,                       /* scroll height */
    0,                       /* scroll length */
    EF_EDITABLE,             /* text can be modified */
    FLAT_BOX,                /* plain border, nothing fancy */
};

```

DeskMate Technical Reference Component Manager

Example 3:

This example creates a 3 line edit field. Each line contains 25 characters. The buffer holds 5 lines of 50 characters each, so the edit field must scroll. Scrolling in this edit field is done automatically by setting `scroll_type` to `EF_NO_SCROLL`.

```
unsigned char sample3_text(50+2)*5+1] = { 0 };
EDITFIELD sample3_edit field[] =
{
    CMP_EDIT,                /* component type */
    ENABLED,                 /* bEnabled */
    0,                       /* xorg */
    0,                       /* yorg */
    25 * CHAR_XEXT,          /* xext (window is 25 chars across) */
    3 * CHAR_YEXT,           /* yext (window is 3 lines high) */
    APP_TAG + CTRL_C,        /* accelerator */
    APP_TAG + CTRL_C,        /* return code */
    EF_WORD_WRAP,            /* edit type */
    EF_NO_HIGHLIGHT,         /* bHighlight is disabled */
    0,                       /* no dec_places, for # fields only */
    0, 0,                   /* origin of buffer */
    50 * CHAR_XEXT,          /* buffer width is 50 characters */
    5 * CHAR_YEXT,           /* buffer is 5 lines high */
    NORMAL,                  /* character attribute */
    0,                       /* position cursor at first character */
    0,                       /* end offset */
    0,                       /* select offset */
    0,                       /* select_length (nothing selected) */
    0,                       /* no terminating character */
    sample3_text,            /* pBuffer */
    0,                       /* no format string */
    EF_NO_SCROLL,            /* i don't scroll, edit field does */
    0,                       /* scroll height */
    0,                       /* scroll length */
    EF_EDITABLE,             /* text can be modified */
    EF_NO_BORDER,            /* no border */
};
```

DeskMate Technical Reference Component Manager

Pre-Defined Components

add_scroll_bar_cmp ()

add_scroll_bar_cmp registers the scroll bar component with the Core Services Resource (CSR).

Input Parameters

None.

Return Value

The component type which will be used by the application to open, draw, run, and close the scroll bar component.

CSR_ERROR if scroll bar could not be added to list.

Special Notes

add_scroll_bar_cmp must be called prior to opening, drawing, running or closing the scroll bar component. The type returned must be placed into the **CMP_HEADER** of the scroll bar structure.

To delete the scroll bar, call **cmp_delete** with the return value from the previously called **add_scroll_bar_cmp**, if not CSR_ERROR.

DeskMate Technical Reference
Component Manager
Pre-Defined Components

cmp_close (Handle, pCMP_HEADER)

int Handle;

CMP_HEADER *pCMP_HEADER;

cmp_close deactivates the specified component.

Input Parameters

Handle is the handle to the component that was assigned when the component was opened.

pCMP_HEADER is a pointer to the component structure, which begins with a **CMP_HEADER**.

Return Value

CSR_ERROR if the component could not be closed, or the **Handle** is incorrect.

Special Notes

After **cmp_close**, **event_read** will no longer detect events on the component. It is the application's responsibility to remove the component image from the screen.

DeskMate Technical Reference Component Manager

Pre-Defined Components

cmp_draw (Handle, pCMP_HEADER)

int Handle;

CMP_HEADER *pCMP_HEADER;

cmp_draw draws the specified component as defined by the *bState* and/or *bEnabled* flags in the component structure.

Input Parameters

Handle is the handle that was assigned to the component when it was opened.

pCMP_HEADER is a pointer to the component structure.

Return Value

<int>

CSR_ERROR if the component could not be drawn.

Edit fields always returns CMP_NO_ACTION.

Special Notes

After **cmp_open** is called, any action on the component will be reported through the event manager.

The component will be drawn as defined by *bState* and *bEnabled*.

LIST BOX specific:

The items in the list box will be drawn as shaded or not shaded as defined by *bEnabled*.

PUSH BUTTON specific:

The push button will appear flat (PB_FLAT) or raised (PB_INV_RAISED) button as specified by the *type* parameter. A flat push button will have a hot spot that is slightly taller than the flat push button image. The hot spot will actually be the size of a raised push button image.

RADIO BUTTON specific:

The group will be drawn in the positions defined by *selected* and each button will be drawn shaded or not shaded as defined by *bEnabled* in each individual radio button definition.

SCROLL BAR specific:

add_scroll_bar_cmp must be called prior to opening, drawing, running or closing the scroll bar component.

DeskMate Technical Reference Component Manager

Pre-Defined Components

cmp_open (pCMP_HEADER)
CMP_HEADER *pCMP_HEADER;

cmp_open draws the component and registers the area of the screen on which the component is drawn with the event manager as a hot spot.

Input Parameters

pCMP_HEADER is a pointer to a component structure.

Return Value

<int>

The handle assigned to the component.

CSR_ERROR if the component could not be opened.

Special Notes

After **cmp_open** is called, any action (mouse or joystick included) on the component will be reported through the event managers **event_read** call.

CHECK BOX specific:

The check box will be drawn as checked or unchecked as defined by **bState** (CB_CHECKED or CB_UNCHECKED).

MENU BAR specific:

It is strongly suggested that when running a menu bar as a component within a dialog box, the message menu and the accessory menu not be displayed.

RADIO BUTTON specific:

The radio button group is defined relative to the frame and the individual buttons are defined relative to the group origin. The shading of the currently pressed radio button will result in an error when the button group is run. The purpose of the x and y dimension definitions is to allow **cmp_run** to intelligently process the arrow keys.

If selected for a radio button group is set to NO_SELECTED, then none of the buttons will be pressed.

See Also

cmp_draw()

DeskMate Technical Reference Component Manager

Pre-Defined Components

cmp_run (Handle, pCMP_HEADER)

int Handle;

CMP_HEADER *pCMP_HEADER;

cmp_run processes the specified component.

Input Parameters

Handle is the handle assigned to the component when it was opened. The **Handle** may be **CSR_ERROR** if the component has not been opened.

pCMP_HEADER is a pointer to a component structure.

Return Value

<int>

CHECK BOX, ICON BUTTON, MENU BAR and PUSH BUTTON return:

CMP_ACTION if either the space bar, a mouse click or pressing the components accelerator caused the component to change state.

CMP_NO_ACTION if TAB, BACKTAB, or an event outside the component caused the return.

EDIT FIELD return:

CMP_ACTION if the buffer was changed.

CMP_GO if the mouse was double-clicked or the **ENTER_KEY** was pressed in single line edit field.

CMP_NO_ACTION if the buffer was not changed.

CMP_SCROLLED indicates the cursor has reached the edge of window and scrolling is needed.

CMP_SELECT_CHANGE indicates when selection changes so the application may update its clipboard items.

LIST BOX return:

CMP_NO_ACTION indicates that either TAB, BACKTAB, an unused arrow, or an event outside the list box caused the routine to return.

CMP_SELECT_CHANGE indicates when the space bar, an arrow key or a mouse click caused a list box state to change.

CMP_GO is returned when the enter key was pressed or there was a double click on an item.

RADIO BUTTON return:

CMP_SELECT_CHANGE indicates when the space bar or the mouse click caused a button to be pushed.

CMP_NO_ACTION indicates that either TAB, BACKTAB, or an event outside the radio button group caused the routine to return.

SCROLL BAR return:

CMP_NO_ACTION if no action taken within the scroll bar.

CMP_SELECT_CHANGE if scroll has occurred.

CMP_APPL if the component received an **EVENT_APPL** event.

All components return:

CMP_DISABLED if the component could not be run because the **bEnabled** flag in the component

DeskMate Technical Reference Component Manager

header was DISABLED.

CSR_ERROR if the component could not be run, or an illegal parameter was passed to `cmp_run`.

Special Notes

Events outside of the component TAB, BACKTAB, a mouse click, striking the space bar or pressing the components accelerator will cause the routine to return to the application. These events are returned the event manager. After the component returns to the application, the application should call `event_read` to determine what event caused it to return, and then take appropriate action.

EDIT FIELD specific:

If the return value is `CMP_ACTION` or `CMP_NO_ACTION`, the event queue will contain the event that caused the edit field to exit (such as an outside event), call `event_read` to process that event.

A select change may occur when a `CMP_GO` event is processed. If edit field returns `CMP_GO`, check `select_length` to update the clipboard items.

MENU BAR specific:

The menu bar component must **NOT** be in the tab order.

PUSH BUTTON specific:

`cmp_run` presses the push button but does not raise the button back up.

A push button which is run in the application's base screen must have its background color match the background color of the main menu bar. This is because of the foreground and background colors of each individual component is determined by the current foreground and background colors of the window rather than a part of the component's structure. The current background color can only be set to a single color, so if the button component is on a different color background than the main menu bar one will be the wrong color. A way to work around this is to create a sub-window and run the button component in that sub-window. In doing so, the sub-window and the base window may have correspondingly different background colors to allow for proper pressing of the buttons. The event manager will automatically switch windows to process the main menu bar if the user accesses it.

A flat push button will have a hot spot that is slightly taller than the flat push button image. The hot spot will actually be the size of a raised push button image.

RADIO BUTTON specific:

The initial cursor blink position is normally the the selected (the pressed) radio button. However, if the selected radio button is disabled (grayed), then the next enabled radio button becomes the initial cursor blink position.

The cursor blink can be moved by either a mouse click or the arrow keys. Note that in multidimensional arrays of buttons there are cases where the keyboard cannot arrow to other enabled radio buttons.

For example:

Enabled	Disabled	Enabled
Disabled	Enabled	Disabled

Both enabled buttons on the top row can be arrow-ed to, but the bottom enabled button cannot.

SCROLL BAR specific:

`add_scroll_bar_cmp` must be called prior to opening, drawing, running or closing the scroll bar component.

DeskMate Technical Reference Component Manager

edt_clear (pEditfield)
EDITFIELD *pEditfield;

edt_clear is used to "clear" the currently selected text in the edit field.

Input Parameters

pEditfield is a pointer to an EDITFIELD structure.

Return Value

None.

Special Notes

edt_clear does not place the selected text on the clipboard. The clipboard is unaffected.

DeskMate Technical Reference Component Manager

edt_copy (pEditfield)
EDITFIELD *pEditfield;

edt_copy is used to "copy" the currently selected text in the edit field to the clipboard.

Input Parameters

pEditfield is a pointer to an EDITFIELD structure.

Return Value

None.

Special Notes

If the edit field contains format characters, the format characters are not copied to the clipboard.

DeskMate Technical Reference Component Manager

edt_cut (pEditfield)
EDITFIELD *pEditfield;

edt_cut is used to "cut" the currently selected portion of the edit field text and place it onto the clipboard.

Input Parameters

pEditfield is a pointer to an EDITFIELD structure.

Return Value

None.

Special Notes

If the edit field contains format characters, these characters are not "cut" to the clipboard and they are not removed from the edit field.

DeskMate Technical Reference Component Manager

edt_format_multi (pEditfield)
EDITFIELD *pEditfield;

edt_format_multi formats the edit field string (for wordwrap) by placing soft carriage returns (0Ah) at the end of each word-wrapped line.

Input Parameters

pEditfield is a pointer to an EDITFIELD structure.

Return Value

None.

Special Notes

edt_format_multi is only for use with multiple line edit fields.

DeskMate Technical Reference Component Manager

edt_form_draw (pForm, pEditfield, char_xext, char_yext)
FORM_HDR *pForm;
EDITFIELD *pEditfield;
char char_xext;
char char_yext;

edt_form_draw adds each line of the edit field's text to the form as an element.

Input Parameters

pForm is a pointer to a FORM_HDR structure.

pEditfield is a pointer to an EDITFIELD structure.

char_xext is the world coordinate width of the characters to be placed on the form.

char_yext is the world coordinate height of the characters to be placed on the form.

Return Value

None.

Special Notes

edt_form_draw assumes the segment registers ds and ss are equal.

See Also

form_add_element() in the Form Manager section.

DeskMate Technical Reference Component Manager

edt_paste (pEditfield)
EDITFIELD *pEditfield;

edt_paste is used to paste the ASCII text that is on the clipboard into the edit field at the current cursor location.

Input Parameters

pEditfield is a pointer to an EDITFIELD structure.

Return Value

CSR_ERROR if the clipboard does not contain data of type CLIP_TEXT, or if the clipboard contains non-numeric data with EF_NUMBER or EF_EXTNUM edit type set. CSR_ERROR is also returned if the edit field's buffer (single and multiple line) is exceeded.
CSR_NULL is returned if successful.

Special Notes

If **edt_paste** is called when text in the edit field is selected, the clipboard text will replace the selected text. If there is selected text, an application should disable their clipboard items before or after **edt_paste** is called. If the clipboard text is too large to fit into the single-line edit field's buffer, the text is truncated. If the multiple line edit field's buffer is exceeded, CSR_ERROR is returned. If pasting to a numeric edit field, the data currently in the edit field buffer will be replaced with the pasted text.

edt_paste uses the format character field to determine the max size to be pasted.

Bugs

If the single line edit field is a formatted, right justified, numeric edit field, CSR_ERROR will **NOT** be returned if the pasted data is too large.

DeskMate Technical Reference Component Manager

edt_replace (pEditfield, lpText, Length)

EDITFIELD *pEditfield;

char far *lpText;

unsigned int Length;

edt_replace replaces the selected portion of a specified edit field with a specified replacement text string.

Input Parameters

pEditfield is a pointer to an EDITFIELD structure.

lpText is a far pointer to a replacement text string.

Length is the length of the replacement string.

Return Value

CSR_ERROR if replacement text does not fit into the specified edit field.

Special Notes

If CSR_ERROR is returned, the edit field will be unmodified.

DeskMate Technical Reference Component Manager

lb_sort_list (pListbox)
LISTBOX *pListbox;

lb_sort_list sorts a list box's items without running or drawing the list box.

Input Parameters

pListbox is a pointer to a LISTBOX structure.

Return Value

None.

Configuration Manager

"Configuration Manager"

Table of Contents

General Description/Notes	5- 1
Structures/Defines	5- 2
cfg_get_com_dataGets port and modem settings.....	5- 6
cfg_get_com_interrupt_flag ..Gets the comm and mouse int flags.	5- 7
cfg_get_com_parmsGets communication parameters.....	5- 8
cfg_get_com_superhayes_flag .Gets the hayes logical flag.....	5- 9
cfg_get_com_timeout_delay ...Gets the communication timeout....	5-10
cfg_get_moreaccsGets "More" menu option status.....	5-11
cfg_get_ms_dataGets mouse configuration data.....	5-12
cfg_get_prt_dataGets current printer data.....	5-13
cfg_get_vid_colorGets palette information.....	5-14
cfg_get_vid_driver ...Gets the video driver name.....	5-16
cfg_get_screen_saver_delay ..Gets the screen saver delay.....	5-17
cfg_reset_temp_config Restores temp configuration settings.....	5-18
cfg_restore_cpiReplaces printer CPI settings	5-19
cfg_set_com_dataSets port and modem settings.....	5-20
cfg_set_com_interrupt_flag ..Sets the comm and mouse int flags.	5-21
cfg_set_com_parmsSets communication parameters.....	5-22
cfg_set_com_superhayes_flag .Sets the fast hayes logical flag...	5-23
cfg_set_com_timeout_delay ...Sets the communication timeout....	5-24
cfg_set_dbl_clickSets the mouse double click speed.....	5-25
cfg_set_moreaccsSets the "More" accessory	5-26
cfg_set_ms_dataSets mouse configuration data.....	5-27
cfg_set_prt_dataSets current printer data.....	5-28
cfg_set_temp_config ..Saves temp configuration settings.....	5-29
cfg_set_temp_cpiTemporarily replaces CPI.....	5-30
cfg_set_vid_colorSets palette information.....	5-31
cfg_set_vid_screen_saver_delay ...Sets screen saver delay.....	5-32
cfg_writeWrites configuration info.....	5-33

General Description/Notes

The configuration routines allow the programmer to get, modify, and save the DeskMate configuration data stored in the DMCSR.CFG configuration data file. The CSR's configuration data is read once during initialization, expanded, and kept in memory during the DeskMate session. The configuration manager will read and write either from EEPROM or disk. Only six (6) colors can be saved to eeprom, but all data can be saved to disk. The majority of the save calls will automatically save the current configuration data to the DMCSR.CFG file, with the exception of the `cfg_set_vid_palette` call which must be followed by a call to `cfg_write` if the colors are to be saved.

DeskMate Technical Reference Configuration Manager

Structures/Defines

```
/*-----*/
/*
/* CSRCFG.H contains the Configuration structures and defines */
/*
/*-----*/

DRIVER_NAME_SIZE 13 /* size of driver names in bytes */

/* Printer configuration */
struct printer_cfg_defn
{
    char    bAutoLF;          /* bAutoLF determines what mode */
                                /* the printer is currently in */
                                /* CR_ONLY or CR_WITH_LF */
    char    port;             /* selected port */
    char    driver[DRIVER_NAME_SIZE]; /* driver filename */
    char    id;               /* driver sub-id */
    char    cpi;              /* characters per inch */
    char    bPause;           /* pause between pages flag */
};
typedef struct printer_cfg_defn PRINTER_CFG;

port:
PRT_LPT1      0
PRT_LPT2      1
PRT_LPT3      2

driver:
The name of the printer driver file name, includes extension.

id:
Internal to DeskMate.

cpi:
PRT_10_CPI    0
PRT_12_CPI    1
PRT_16_CPI    2

bPause:
ENABLED/DISABLED

/* Mouse */
struct mouse_cfg_defn
{
    char    bSelected;        /* indicates if user wants a mouse */
    char    click_speed;      /* double click timeout value */
    char    port;             /* mouse driver port */
    char    driver[DRIVER_NAME_SIZE]; /* filename of */
                                    /* the mouse driver */
};
typedef struct mouse_cfg_defn MOUSE_CFG;
```

DeskMate Technical Reference Configuration Manager

bSelected:
SELECTED/DESELECTED

click_speed:
MS_MAX_CLICK_SPEED 4

port:
MS_COMPORT1 0
MS_COMPORT2 1

driver:
The name of the mouse driver including extension. For example DMMDSERI.RES.

```
/* Communications */
struct cparms_defn
{
    char    bXon;           /* Xon/Xoff enable flag */
    char    bASCFilter;     /* ASCII filter enable flag */
    char    bLFfilter;      /* line filter enable flag */
    char    bStatus;        /* com port status flag (OPEN or CLOSED) */
    char    bCarrier;       /* carrier det stat flag (CARRY/NO_CARRY) */
    int     baud_rate;      /* line speed setting */
    char    word_size;      /* 7 or 8 */
    char    parity;         /* EVEN_PARITY, ODD_PARITY or NO_PARITY */
    char    stop_bits;      /* 1 or 2 */
};
typedef struct cparms_defn CPARMS;

struct com_buffer_info_defn
{
    char far *rs232_start_ptr;
    char far *rs232_end_ptr;
    char far *input_ptr;
    char far *output_ptr;
    int     rs232_buf_size;
    int     count_num;
    char    xon_xoff_flag;
};
typedef struct com_buffer_info_defn COM_BUFFER_INFO;
struct com_state_defn
{
    char    com_port;       /* COM1 or COM2 */
    char    line_type1;     /* DIRECT or MODEM */
    char    modem1;         /* modem name index */
    char    line_type2;     /* DIRECT or MODEM */
    char    modem2;         /* modem name index */
    char    line_type3;     /* DIRECT or MODEM */
    char    modem3;         /* modem name index */
    char    line_type4;     /* DIRECT or MODEM */
    char    modem4;         /* modem name index */
};
typedef struct com_state_defn COM_STATE;
```

DeskMate Technical Reference Configuration Manager

```
/* Parity settings */
NO_PARITY      0
ODD_PARITY     1
EVEN_PARITY    2
```

```
/* Port addresses */
COM1           0
COM2           1
COM3           2
COM4           3
```

```
/* Line types */
MODEM          1          /* modem is attached to line */
DIRECT         0          /* direct connect line */
```

```
/* Port status */
PORT_OPEN      1          /* terminal active - dtr and rts */
PORT_CLOSED    0          /* terminal not ready - no dtr or rts */
```

```
/* Carrier state */
CARRY          1
NO_CARRY       0
```

```
/* Help Level values */
CFG_NEW_USER   0
CFG_BEGINNING_USER 1
CFG_INTERMEDIATE_USER 2
CFG_EXPERT_USER 3
```

```
/*-----*/
/*                                           */
/* CSRVID.H contains the VID_PALETTE structure */
/*                                           */
/*-----*/
```

The VID_PALETTE structure is defined as follows:

```
struct vid_palette_defn
{
```

```
    char palette;    /* This element is the index
of the CSR configuration palette to get. This
index should be 0-15 as only sixteen palette
settings are maintained within the csr
configuration. */
```

```
    char red;        /* This element is the red gun
setting within the virtual range of 0-192. All
DeskMate video drivers will interpret this value
relative to the intensity resolution for each
color gun of that particular video device. 0 is
the lowest intensity setting across all devices
and 192 is the highest intensity setting across
all devices. See vid_set_palette for some
example color settings. */
```

```
    char green;      /* This element is the green gun setting. */
```


DeskMate Technical Reference Configuration Manager

```
char blue;          /* This element is the blue gun setting. */  
};  
typedef struct vid_palette_defn VID_PALETTE;
```

palette:

The index of the color palette to be selected from the current CSR configuration. The CSR configuration maintains 16 settings. Use a value from 0 to 15 to access the desired palette.

red:

The value assigned controls the intensity of the red video gun. Any value in the range 0 to 192 is acceptable; 0 produces minimum intensity, 192 produces maximum intensity. DeskMate video drivers interpret this value relative to the intensity resolution for each color gun of that particular video device.

green:

The value assigned to this element controls the intensity of the green video gun. See Red (above) for more information.

blue:

The value assigned to this element controls the intensity of the blue video gun. See Red (above) for more information.

DeskMate Technical Reference Configuration Manager

cfg_get_com_data (pCOM_STATE)
COM_STATE *pCOM_STATE;

cfg_get_com_data gets the current port and modem communication configuration settings.

Input Parameters

pCOM_STATE is a pointer to a COM_STATE structure in which to store the communication settings.

Return Value

None.

See Also

COM_STATE structure.

DeskMate Technical Reference Configuration Manager

cfg_get_com_interrupt_flag ()

cfg_get_com_interrupt_flag retrieves the communication and mouse port interrupt flags. These flags indicate whether the high or low interrupts are being used.

Input Parameters

None.

Return Value

<int>

Comm interrupt flags. The least significant bit of the flag corresponds to the communications port, while the next bit corresponds to the mouse port. A bit value of zero (0) indicates the low interrupt is to be used, while a one (1) indicates the high interrupt is to be used.

Special Notes

cfg_get_com_interrupt_flag is only available when running DeskMate 03.05 and later.

See Also

cfg_set_com_interrupt_flag()

DeskMate Technical Reference Configuration Manager

cfg_get_com_parms (pCPARMS)
CPARMS *pCPARMS;

cfg_get_com_parms gets the current communication parameter settings.

Input Parameters

pCparms is a pointer to a CPARMS structure.

Return Value

None.

Special Notes

The settings are returned in the CPARMS structure.

See Also

CPARMS structure.

DeskMate Technical Reference Configuration Manager

cfg_get_com_superhayes_flag ()

cfg_get_com_superhayes_flag retrieves the fast hayes logical flag.

Input Parameters

None.

Return Value

<int>

SELECTED if the flag is set.

DESELECTED if the flag is cleared.

Special Notes

cfg_get_com_superhayes_flag is only available when running DeskMate 03.05 and later.

The fast hayes logical flag indicates a hayes modem that does not require the full Hayes standard timeouts and therefore, can operate faster for dialing and disconnecting. **cfg_get_com_superhayes_flag** is used by an application to determine if the user has selected this type of modem.

DeskMate Technical Reference Configuration Manager

cfg_get_com_timeout_delay ()

cfg_get_com_timeout_delay retrieves the communications timeout delay. This value is used by **com_call** to determine when to terminate an unanswered call.

Input Parameters

None.

Return Value

<int>

Comm timeout delay in seconds.

Special Notes

cfg_get_com_timeout_delay is only available when running DeskMate 03.05 and later.

DeskMate Technical Reference Configuration Manager

cfg_get_moreaccs ()

cfg_get_moreaccs used to determine if the "More" accessory menu option exists.

Input Parameters

None.

Return Value

<int>

SELECTED if the "More" accessories exist.

DESELECTED if the "More" accessory does not exist.

Special Notes

cfg_get_moreaccs is only available when running DeskMate 03.03 and later.

The determination is made by reading a flag in the configuration file.

DeskMate Technical Reference Configuration Manager

```
cfg_get_ms_data ( pMOUSE_CFG )  
MOUSE_CFG *pMOUSE_CFG;
```

cfg_get_ms_data gets the current mouse configuration data.

Input Parameters

pMOUSE_CFG is a pointer to a MOUSE_CFG structure in which to store the mouse configuration data.

Return Value

None.

See Also

MOUSE_CFG structure.

DeskMate Technical Reference Configuration Manager

cfg_get_prt_data (pPRINTER_CFG)
PRINTER_CFG *pPRINTER_CFG;

cfg_get_prt_data gets the current printer driver configuration data.

Input Parameters

pPRINTER_CFG is a pointer to a PRINTER_CFG structure in which to store the printer driver configuration data.

Return Value

None.

Example

```
/* check to see if there is a printer driver */
PRINTER_CFG PrtConfig;

cfg_get_prt_data( &PrtConfig );
if ( PrtConfig.driver[0] == '/' )
    /* no printer driver */
```

See Also

PRINTER_CFG structure.

DeskMate Technical Reference Configuration Manager

cfg_get_vid_color (pVID_PALETTE)
VID_PALETTE *pVID_PALETTE;

cfg_get_vid_color gets the CSR configuration palette setting for the specified palette index.

Input Parameters

pVID_PALETTE is a pointer to a **VID_PALETTE** structure.

Return Value

CSR_ERROR if the specified palette is greater than the maximum supported palette (0 - 15).
Zero (0) if a valid palette number.

Special Notes

cfg_get_vid_color gets the palette within the CSR configuration and is not necessarily the current palette setting for the video device.

Example

Black (no color):

red = 0;
green = 0;
blue = 0;

Medium Gray (medium intensity white):

red = 128;
green = 128;
blue = 128;

White (highest intensity):

red = 192;
green = 192;
blue = 192;

Dark Red (low intensity red):

red = 64;
green = 0;
blue = 0;

Bright Green (highest intensity green):

red = 0;
green = 192;
blue = 0;

Medium Magenta (medium intensity):

red = 127;
green = 0;
blue = 128;

DeskMate Technical Reference Configuration Manager

See Also

`vid_set_palette()`

`vid_get_palette()`

The `VID_PALETTE` structure as defined in `CSRVID.H`.

DeskMate Technical Reference Configuration Manager

cfg_get_vid_driver (pVideoDriver)
char *pVideoDriver[DRIVER_NAME_SIZE];

cfg_get_vid_driver gets the string of the current video driver name.

Input Parameters

pVideoDriver is a pointer to a string of **DRIVER_NAME_SIZE** characters long in which to store the video driver name string.

Return Value

None.

Special Notes

If a null string is returned in **pVideoDriver** then AUTO DETECTION of the video driver is enabled.

See Also

vid_inquire_device()

DeskMate Technical Reference Configuration Manager

cfg_get_vid_screen_saver_delay ()

cfg_get_screen_saver_delay retrieves the screen saver delay index.

Input Parameters

None.

Return Value

<int>

Screen saver delay index.

This index controls the ability of the video drivers to blank the screen after a period of keyboard or mouse inactivity.

Special Notes

The screen saver delay index may be interpreted as follows:

- 0 = No screen blanking
- 1 = 5 minutes
- 2 = 15 minutes
- 3 = 30 minutes
- 4 = 1 hour
- 5 = 2 hours
- 6 = 4 hours
- 7 = 8 hours.

cfg_get_vid_screen_saver_delay is only available when running DeskMate 03.05 and later.

DeskMate Technical Reference Configuration Manager

cfg_reset_temp_config ()

cfg_reset_temp_config restores the configuration settings saved through **cfg_set_temp_config** thus cancelling the write protect action.

Input Parameters

None.

Return Value

None.

Special Notes

cfg_reset_temp_config is only available when running DeskMate 03.03 and later.

See Also

cfg_set_temp_config()

DeskMate Technical Reference Configuration Manager

cfg_restore_cpi ()

cfg_restore_cpi replaces the printer character-per-inch setting that existed before the first cfg_set_temp_cpi call.

Input Parameters

None.

Return Value

None.

Special Notes

cfg_restore_cpi is only available when running DeskMate 03.03 and later.

See Also

cfg_set_temp_cpi()

DeskMate Technical Reference Configuration Manager

cfg_set_com_data (pCOM_STATE)
COM_STATE *pCOM_STATE;

cfg_set_com_data sets the port and modem communication configuration data.

Input Parameters

pCOM_STATE is a pointer to a COM_STATE structure which contains the desired port and modem communication configuration data.

Return Value

CSR_ERROR if the configuration was not written.

Special Notes

If the data has changed the entire configuration will be written to EEPROM or disk.

See Also

COM_STATE structure.

DeskMate Technical Reference Configuration Manager

cfg_set_com_interrupt_flag (Flags)

Int Flags;

cfg_set_com_interrupt_flag sets the communications and mouse interrupt flags.

Input Parameters

Flags contains the new comm flag values to use. The flag may be set from 0 to 3. The least significant bit corresponds to the communications port, while the next bit corresponds to the mouse port. A bit value of zero (0) indicates the low interrupt is to be used, while a one (1) indicates the high interrupt is to be used. For example:

PORT	LOW	HIGH
COM3	IRQ2	IRQ4
COM4	IRQ3	IRQ5

0 = low IRQ; 1 = high IRQ

For the serial port: bit 0 "xxxx xxxx xxxx xxxC"

For the serial mouse: bit 1 "xxxx xxxx xxxx xxMx"

BIT 0

If COM3 selected:

0 = communication on IRQ2

1 = communication on IRQ4

If COM4 selected:

0 = communication on IRQ3

1 = communication on IRQ5

BIT 1

If COM3 selected:

0 = mouse on IRQ2

1 = mouse on IRQ4

If COM4 selected:

0 = mouse on IRQ3

1 = mouse on IRQ5

Return Value

None.

Special Notes

cfg_set_com_interrupt_flag is only available when running DeskMate 03.05 and later.

To select COM3 and COM4 use **cfg_set_com_data**.

DeskMate Technical Reference Configuration Manager

cfg_set_com_parms (pCPARMS)
CPARMS *pCPARMS;

cfg_set_com_parms sets the communication parameter configuration data.

Input Parameters

pCPARMS is a pointer to a CPARMS structure which contains the desired parameter configuration.

Return Value

CSR_ERROR if the configuration was not written.

Special Notes

If the data has changed the entire configuration will be written to EEPROM or disk.

See Also

CPARMS structure.

DeskMate Technical Reference Configuration Manager

cfg_set_com_superhayes_flag (Value)

Int Value;

cfg_set_com_superhayes_flag sets the fast hayes logical flag.

Input Parameters

Value is the new flag value to use. This value should be SELECTED or DESELECTED.

Return Value

None.

Special Notes

cfg_set_com_superhayes_flag is only available when running DeskMate 03.05 and later.

The fast hayes logical flag indicates a hayes modem that does not require the full Hayes standard timeouts and therefore, can operate faster for dialing and disconnecting.

DeskMate Technical Reference Configuration Manager

cfg_set_com_timeout_delay (DelaySeconds)
Int DelaySeconds;

cfg_set_com_timeout_delay sets the communications timeout delay in seconds.

Input Parameters

DelaySeconds is the new delay value to use. This value may range from 1 thru 127 seconds.

Return Value

None.

Special Notes

A value of 0 should not be used, as this will cause immediate termination of a call.

cfg_set_com_timeout_delay is only available when running DeskMate 03.05 and later.

DeskMate Technical Reference Configuration Manager

cfg_set_dbl_click (DoubleClickSpeed)

int DoubleClickSpeed;

cfg_set_dbl_click temporarily sets the mouse double click speed.

Input Parameters

DoubleClickSpeed is an integer value from zero (0) to four (4).

Return Value

None.

Special Notes

The click speed will **NOT** be written to EEPROM or disk, as this is used for a temporary setting. Zero requires a faster click speed, four requires a slower click speed.

DeskMate Technical Reference Configuration Manager

cfg_set_moreaccs (bAction)

int bAction;

cfg_set_moreaccs sets the "More" accessory flag in the configuration file, and calls **cfg_write** to record that change.

Input Parameters

bAction is **SELECTED** if the "More" accessory is to be displayed; **DESELECTED** if the "More" accessory is not to be displayed.

Return Value

None.

Special Notes

cfg_set_moreaccs is only available when running DeskMate 03.03 and later.

DeskMate Technical Reference Configuration Manager

cfg_set_ms_data (pMOUSE_CFG)
MOUSE_CFG *pMMOUSE_CFG;

cfg_set_ms_data sets the mouse configuration.

Input Parameters

pMOUSE_CFG is a pointer to a **MOUSE_CFG** structure which contains the desired mouse configuration.

Return Value

CSR_ERROR if the configuration could not be set.

Special Notes

If the data has not changed no action is taken. If the data has changed the mouse driver is unloaded, (if loaded), if **bSelected** is **DISABLED** the new mouse driver is not loaded, if **bSelected** is **ENABLED** the new mouse driver is loaded with the new configuration information, and a call is made to **cfg_write** to save the entire configuration information to EEPROM or disk.

Following a call to **cfg_set_ms_data** the 9th byte of the sturcture passed to it will always be zero. This is the 6th byte of the mouse driver name. Since all mouse drivers are now in the form of **DMMDX.RES**, the set routine forces a terminating zero in the 6th character of the string. This allows old applications, which my try to load **DMMDSERI.RES** or **DMMDJOY.RES**, to load the correct driver.

DeskMate Technical Reference Configuration Manager

```
cfg_set_prt_data ( pPRINTER_CFG )  
PRINTER_CFG *pPRINTER_CFG;
```

cfg_set_prt_data sets the printer data configuration.

Input Parameters

pPRINTER_CFG is a pointer to a **PRINTER_CFG** structure which contains the desired printer configuration.

Return Value

CSR_ERROR if the configuration is not set.

Special Notes

The entire configuration will be written to EEPROM or disk.

Example

```
PRINTER_CFG      PrtConfig;  
  
int   ReturnCode;  
ReturnCode = cfg_set_prt_data( &PrtConfig );  
/* transfer configuration data to DeskMate internals */  
prt_get_printer( &PrinterData );
```


DeskMate Technical Reference Configuration Manager

cfg_set_temp_config ()

cfg_set_temp_config saves the current configuration settings to a buffer, and writes protects the configuration file.

Input Parameters

None.

Return Value

None.

Special Notes

cfg_set_temp_config is only available when running DeskMate 03.03 and later.

See Also

cfg_reset_temp_config()

DeskMate Technical Reference Configuration Manager

cfg_set_temp_cpi (NewCPI)
int NewCPI;

cfg_set_temp_cpi temporarily replaces the current printer parameter, character-per-inch, with the supplied value.

Input Parameters

NewCPI the new CPI value to be used. The values should be **PRT_10_CPI**, **PRT_12_CPI** or **PRT_16_CPI**, as defined in **CSRCFG.H**.

Return Value

None.

Special Notes

cfg_set_temp_cpi is only available when running DeskMate 03.03 and later.

DeskMate Technical Reference Configuration Manager

cfg_set_vid_color (pVID_PALETTE)
VID_PALETTE *pVID_PALETTE;

cfg_set_vid_color sets the CSR configuration palette setting for the specified palette index.

Input Parameters

pPalette is a pointer to a VID_PALETTE structure.

Return Value

CSR_ERROR if the maximum palette index is exceeded.

Special Notes

cfg_set_vid_color sets the palette within the CSR configuration, but does not change the color on the screen.

Example

Black (no color):

```
red = 0;  
green = 0;  
blue = 0;
```

Medium Gray (medium intensity white):

```
red = 128;  
green = 128;  
blue = 128;
```

White (highest intensity):

```
red = 192;  
green = 192;  
blue = 192;
```

Dark Red (low intensity red):

```
red = 64;  
green = 0;  
blue = 0;
```

Bright Green (highest intensity green):

```
red = 0;  
green = 192;  
blue = 0;
```

Medium Magenta (medium intensity):

```
red = 128;  
green = 0;  
blue = 128;
```

See Also

VID_PALETTE structure as defined in CSRVID.H

DeskMate Technical Reference Configuration Manager

cfg_set_vid_screen_saver_delay (DelayIndex)

int DelayIndex;

cfg_set_screen_saver_delay sets the screen saver delay index. This index controls the ability of the video drivers to blank the screen after a period of keyboard or mouse inactivity.

Input Parameters

DelayIndex is the new delay index to use. The index may be set from 0 thru 7, corresponding to the following delays:

- 0 = No screen blanking
- 1 = 5 minutes
- 2 = 15 minutes
- 3 = 30 minutes
- 4 = 1 hour
- 5 = 2 hours
- 6 = 4 hours
- 7 = 8 hours.

Return Value

None.

Special Notes

cfg_set_vid_screen_saver_delay is only available when running DeskMate 03.05 and later.

DeskMate Technical Reference Configuration Manager

cfg_write ()

cfg_write writes the entire CSR configuration to EEPROM or disk. The exception is that if **cfg_set_temp_config** has been called, the CSR configuration information will **NOT** be written to disk or EEPROM.

Input Parameters

None.

Return Value

CSR_ERROR if the write was unsuccessful.

Special Notes

cfg_write is provided to allow all color palettes to be set before writing to EEPROM or disk.

See Also

cfg_reset_temp_config()
cfg_set_temp_config()

Database Resource

"Database Manager"

Table of Contents

General Description/Notes	6- 1
Structures/Defines	6- 3
ADD_ROW	Adds a row of data to specified table..... 6-13
ADD_ROW_NO_FLUSH ..	Adds a row to specified table w/no write.... 6-14
ALTER_TABLE	Alters the definition of a table..... 6-15
CLOSE_FILE	Closes a file..... 6-16
CLOSE_TABLE	Closes a table..... 6-17
COUNT_RECORDS	Counts the number of records..... 6-18
COPY_LAYOUTS	Copies selected tables to new file..... 6-19
CREATE_FILE	Creates a new file..... 6-20
CREATE_TABLE	Creates a new table within a file..... 6-21
db_bind_end	Terminates the bindings to DB manager..... 6-22
db_bind_init	Loads the DB manager into memory..... 6-23
db_bind_read	Binds to 1989 Database Read Resource..... 6-24
db_end_read	Terminate bind to 1989 DB Read Resource..... 6-25
DEFINE_INDEX	Defines the sort order..... 6-26
DELETE_MULTI_ROWS ..	Deletes all rows in a specified table..... 6-27
DELETE_ROW	Deletes a specified row..... 6-28
DROP_INDEX	Drops an index from a table..... 6-29
DROP_TABLE	Drops a table and data from file..... 6-30
END_DB.....	Unbinds 1989 DB resources from Manager..... 6-31
END_TEMP_SORT	Ends the temporary sort order..... 6-32
FETCH_RECORD	Fetch a record by record number..... 6-33
FIRST_RECORD	Fetches a single record at a time..... 6-34
GET_COL_INFO	Gets column information..... 6-35
GET_COLUMN_NAMES ..	Gets the column names in a table..... 6-36
GET_INDEX_INFO	Gets the index information for a table..... 6-37
GET_MAX	Gets the highest value for column name..... 6-38
GET_MIN	Gets the lowest value for column name..... 6-39
GET_PAGE_SETUP	Gets the page setup information..... 6-40
GET_RECORD_LOCATION ..	Gets previous and next record numbers..... 6-41
GET_TABLE_NAMES	Gets the table names in a file..... 6-42
INIT_BD.....	Binds 1989 DB resources to Manager..... 6-43

LAST_RECORD	Fetches a single record at a time.....	6-44
MERGE	Merges two tables together.....	6-45
MORE_RECORDS	Gets more records which match query.....	6-46
NEXT_RECORD	Fetches a single record at a time.....	6-48
OPEN_FILE	Opens a file.....	6-49
OPEN_TABLE	Opens a table.....	6-50
PACK_TABLE	Packs a table/Removes deleted records.....	6-52
PREV_RECORD	Fetches a single record at a time.....	6-53
SAVE_PAGE_SETUP	Saves the page setup information.....	6-54
SETUP_QUERY	Sets up query constraints.....	6-55
TEMP_SORT	Defines a temporary sort order.....	6-58
UPDATE_ROW	Updates the row specified.....	6-59
Error Code List		6-60

General Description/Notes

The database manager resource provides record level access into files both on and off the network for applications. The database will not be linked with the application, but will be a service resource. It will service one request at a time and will handle multiple users. A database file is made up of one or more tables. Each table describes a record layout containing one or more fields. A column in a table refers to a field in a record. A row in a table refers to a record. A data dictionary is maintained to store information about the tables and the columns belonging to those tables. There are three types of data that can be defined: Character, Numeric, or Date, although all data is stored on disk in ASCII format. Date fields are stored in Julian format to provide for proper sorting.

Applications provide the definition of records in a file, and the database supports functions such as adding, deleting, sorting, and searching. The size of database files is limited to 348,160 bytes for the DeskMate 03.02 product. The workgroup companion database which is on a server machine in a network environment will allow a database file up to 675,840 bytes. Applications using the database must manage all user interaction and error handling. The calls which utilize the workgroup companion are available upon request from DeskMate Support Services.

The following interface routines will be allowed through a direct link or via desk or across a network. Access to the database manager will be provided through a single entry point. To utilize these routines the application will make the following call using the appropriate interface routine name and parameter. These parameters are found in DMDB.H.

```
erc = db_mgr( INTERFACE_ROUTINE, PARAMETER );
```

For example to open a table :

```
erc = db_mgr( OPEN_TABLE, pTableInfo );
```

The DeskMate 03.03 and later database resource is made up of three separate pieces. Below is a list of the different resources and the calls contained in those resources.

Read Resource

CLOSE_FILE	CLOSE_TABLE	COUNT_RECORDS
END_DB	FETCH_RECORD	FIRST_RECORD
GET_COL_INFO	GET_COLUMN_NAMES	GET_INDEX_INFO
GET_MAX	GET_MIN	GET_PAGE_SETUP
GET_RECORD_LOCATION	GET_TABLE_NAMES	INIT_DB
LAST_RECORD	MORE_RECORDS	NEXT_RECORD
OPEN_FILE	OPEN_TABLE	PREV_RECORD
SETUP_QUERY		

Update Resource

ADD_ROW	ADD_ROW_NO_FLUSH	DELETE_MULTI_ROWS
DELETE_ROW	END_TEMP_SORT	MERGE
SAVE_PAGE_SETUP	TEMP_SORT	UPDATE_ROW

DeskMate Technical Reference Database Resource

Build Resource

ALTER_TABLE
CREATE_TABLE
DROP_TABLE

COPY_LAYOUTS
DEFINE_INDEX
PACK_TABLE

CREATE_FILE
DROP_INDEX

In DeskMate 03.03 two new bindings were created to allow for the new resources. They are **db_bind_read** and **db_end_read**. These two new binding calls only bind to the read resource. In DeskMate 03.02.0x **db_bind_init** and **db_bind_end** calls were used to bind to all of the Database resources. If an application calls **db_bind_read**, it should not call **db_bind_init**. If an application, which links with the 03.03 and later libraries, is running in a 03.02 environment, when the call **db_bind_read** is made, all of the database resources are loaded.

For the resources DBREAD.RES, DBUPDATE.RES versions less than or equal to 100, and for DBBUILD.RES versions less than or equal to 110, the following problem exists. On a floppy system or in a network environment, the functions **CREATE_FILE**, **CREATE_TABLE**, **COPY_LAYOUTS**, **PACK_TABLE**, **SAVE_PAGE_SETUP**, and **MERGE** do not flush the file buffers. If the disk is removed, the replacement disk will be corrupted as soon as DOS flushes it's buffers. There are two work arounds for this problem. The work around when first creating a file is to either create and index with **DEFINE_INDEX** which will flush the buffers, or if no index is desired then call **DROP_INDEX** which will flush the buffers. The second work around, which is more generic, involves calling **DELETE_ROW**. When calling **DELETE_ROW** pass a valid **table_handle**, set **num_query_lines** to one (**num_query_lines = 1**), and for **query_line_array** pass a pointer to a pointer to an invalid string, such as "***". **DELETE_ROW** will return **DB_QUERY_SYNTAX**, this error can be ignored, and the buffers will be flushed.

Records returned by the various calls are of variable length. Records are returned as a sequence of null-terminated strings. Following the final null is an ER_TKN (2).

DeskMate Technical Reference

Database Resource

Structures/Defines

```

/*-----*/
/*
/* DMDB.H contains the Database structures and defines */
/*-----*/

/* defines for dbmgr main case statement */
CREATE FILE      1
OPEN FILE        2
CLOSE FILE       3
CREATE TABLE    4
ALTER TABLE     5
OPEN TABLE      6
ADD ROW          7
DROP TABLE      8
CLOSE TABLE     9
COUNT RECORDS  10
FIRST RECORD     11
LAST RECORD      12
PREV RECORD      13
NEXT RECORD      14
SETUP QUERY      15
MORE RECORDS     16
FETCH RECORD     17
DELETE ROW       18
UPDATE ROW       19
GET TABLE NAMES      20
GET COLUMN NAMES      21
COPY LAYOUTS          22
PACK TABLE           23
DEFINE INDEX           24
GET PAGE SETUP        25
SAVE PAGE SETUP       26
DROP INDEX            27
DELETE MULTI ROWS     28
GET INDEX INFO        29
MERGE                 30
GET MIN               31
GET MAX               32
GET COL INFO          33
TEMP SORT             36
END TEMP SORT         37
GET RECORD LOCATION   38
ADD ROW NO FLUSH      39
DB PAUSE              50
INIT DB               51
END DB                52

/* defines for DB PAUSE */
PAUSE STATUS 0
STOP DB      1
START_DB     2
/* Return the status */
/* Stop the DBMGR */
/* Start the DBMGR. */

/* defines for initialization binding and unbinding */
BIND_READ_ONLY 0
BIND_ALL        1
BIND_UPDATE     2
BIND_BUILD      3
BIND_NETWK      4
READ_ONLY       5
/* for 89 app support */
/* for server/88 network app support */
/* bind update resource */
/* bind build and update resources */
/* bind network resource */
/* parameter to db_bind_read */

```

DeskMate Technical Reference Database Resource

```

READ_WRITE      6                /* parameter to db_bind_read */

MAX_REC_SIZE    1020 /* PG SIZE - TOKEN OVERHEAD - RECORD_OVERHEAD */
ER_TKN         0x02 /* end of record token */
KS_TKN         0x04 /* separator for pSortOrder */

LOWEST_VALID_CHAR 0x07 /* smallest character allowed */
                  /* for app data */

MAX_HIST_SLOTS  10 /* maximum partial update slots */

/* internal database limits */
MTABS           20 /* maximum number of tables in a database */
MCOLS           40 /* maximum columns allowed in a table */
MQLINES         12 /* maximum number of query lines */
MSORT_FLDS      5  /* maximum number of sort fields */
MAX_COL_NAME    20 /* maximum number of chars in column name */
MAX_TAB_NAME    20 /* maximum number of chars in table name */
MAX_QLINE_LEN   62 /* maximum number of chars in query line */

/* error messages from db mgr */
/* (See Appendix A - for explanations) */
DB_OK           0
DB_NO_TBL_HIST  -1
DB_ALREADY_OPEN -2
DB_DISK_WRITE   -3
DB_NO_CREATE    -4
DB_DUPL_TABLE    -5
DB_TBL_NAME_ERR -6
DB_COL_NAME_ERR -7
DB_FIRST_RECORD -8
DB_LAST_RECORD  -9
DB_QUERY_SYNTAX -10
DB_NO_ROWS_SELECTED -11
DB_DISK_READ    -12
DB_NO_OPEN      -13
DB_DUPL_COLUMN  -14
DB_RECNUM_ERR   -15
DB_OUT_OF_MEMORY -16
DB_MAX_NBR_FIELDS -17
DB_INVALID_FILE_TYPE -18
DB_MULTI_UNIQ   -19
DB_UNIQ_NOT_SORT -20
DB_INVALID_CHAR -21
DB_NO_EOR       -22
DB_UNIMPLEMENTED -23
DB_RECORD_NOT_CURRENT -24
DB_TOO_MANY_OPEN_FILES -25
DB_INVALID_FILE_HANDLE -26
DB_TABLE_LOCKED -27
DB_DUPL_KEY_FIELD -28
DB_TABLE_SHARED -29
DB_INVALID_TABLE_HANDLE -30
DB_RECORD_DELETED -31
DB_NO_DBCOLUMNS -32
DB_NO_TEMP_SORT -33
DB_NO_PACK      -34
DB_PAGE_NOT_FOUND -35
DB_NO_RECORDS   -36
DB_DIR_FULL     -37
DB_NO_INDEX     -38
DB_ONLY_RECORD  -39
DB_SAME_TABLE    -40
DB_DIFFERENT_TABLES -41

```

The maximum database record size is 1020 characters. There is no maximum character field within this overall limit. The maximum size for a numeric field is 39 characters.

DeskMate Technical Reference Database Resource

```

DB_DISK_FULL          -42
DB_STOPPED            -43
DB_WRITE_PROTECTED    -44
DB_TABLE_FULL         -45
DB_TABLE_NEEDS_UPDATE -46
DB_MULTI_AGAIN        -49
DB_BUSY               -50

DB_TIMEOUT            -51
DB_SERVER_BUSY        -52
DB_NO_SERVER          -53
DB_SERVER_DIED        -54
DB_NETERR             -55

DB_TEMP_SORT_ALREADY -56      /* may be temp error */

DB_GOT_MINIMUM        -57
DB_READ_ONLY          -58
DB_RESOURCE_NOT_LOADED -59
DB_NEEDS_PACK         -60
DB_NO_FLUSH           -61

MAX_ERR_MSG_NUM       61

/* defines for client calls */
CL_LOCAL              0      /* not a client */
CL_CLIENT             1      /* client on non-server */
CL_RESUME             2      /* resume client on non-server */
CL_REGISTER           3      /* register client with server */
                        /* closes all files and tables */
CL_SRVR               -1     /* client on server */
CL_PAUSE              -2     /* pause client on non-server */

/* structures for database manager calls */
typedef unsigned int UNSIGNED;      /* used on all rec_num */
                                   /* elements */

struct db_columnsx      /* used internally as */
                        /* part of db_table */
{
    char *col_name      /* pointer to column name */
    char *new_name;     /* used to change col name */
    int  col_length;    /* column length */
    char col_type;       /* column type 'C' = char */
                        /* 'N'=numeric */
    char col_attr;      /* column protection attr */
    char unique_flag;   /* non-zero column value */
                        /* must be unique */
};
typedef struct db_columnsx db_columns;

Utilized by:
    db_table structure;

/* defines for col type */
CHAR87  'C'  /* 1987 compatible - to upper &0x5F */
CHAR88  'K'  /* for DOS 3.3 & greater uses */
          /* collating sequence table for */
          /* sorting & queries */

NUMERIC_COL  'N'
DATE_COL     'D'      /* stored internally in julian format */

```

DeskMate Technical Reference Database Resource

```

/* defines for setting unique_flag value */
SET UNIQUE_COL      '1'
UNSET_UNIQUE_COL    '0'

struct db_tablex      /* used in CREATE TABLE, */
                      /* ALTER_TABLE */
{
    int    handle;      /* handle of open file(create) */
                      /* or table(alter) */
    char  *tbl_name;    /* pointer to table name */
    int   n_columns;    /* number of columns in table */
    char  update_type;  /* update function number */
    int   n_items;      /* number of items in the */
                      /* db_column array */
    db_columns *cols;   /* ptr to array of db_column*/
                      /* structures */
};
typedef struct db_tablex db_table;

Utilized by:
    CREATE_TABLE;
    ALTER_TABLE;

/* defines for ALTER_TABLE update_type */
ADD_COLUMN      1
DROP_COLUMN     2
CHG_COLUMN      3
COLUMN_INFO     4

struct table_accessx /* used in OPEN_TABLE, DROP_TABLE */
{
    int    file_handle; /* handle of open file */
    char  *tbl_name;    /* table name to open */
    int   access_level; /* DEFINITION or DATA_ACCESS */
};
typedef struct table_accessx table_access;

Utilized by:
    OPEN_TABLE;
    DROP_TABLE;

/* defines for OPEN_TABLE access_level */
DEFINITION     -2
DATA_ACCESS     1

struct db_valuex      /* used internally in db_add structure*/
{
    char  *col_name;    /* pointer to column name */
    char  *col_value;   /* pointer to column value */
};
typedef struct db_valuex db_value;

Utilized by:
    db_add structure;

struct db_addx        /* used in ADD_ROW, UPDATE_ROW */
                      /* single user calls */
{
    int    table_handle; /* handle of open table */
    int    n_columns;    /* number of columns to be */
                      /* added/updated */
    db_value *val;       /* ptr to an array of db_value */
                      /* structs */
}

```

DeskMate Technical Reference Database Resource

```

    int      prev_rec;    /* previous record, zero means */
                          /* first record */
    int      next_rec;    /* next record, 0= last record */
};
typedef struct db_addx db_add;

Utilized by:
    ADD_ROW
    UPDATE_ROW

struct db_mergex
{
    int      to_th;        /* handle of table merging to */
    int      from_th;      /* handle of table merging from */
    char     duplicates;   /* DUP - Allow duplicate records */
                          /* NO_DUP-Don't allow duplicates */
};
typedef struct db_mergex db_merge;

/* defines for duplicates */
DUP      1
NO_DUP   0

Utilized by:
    MERGE

```

Notes

to_th:

Must be open for DEFINITION.

from_th:

May be open either DEFINITION or DATA_ACCESS.

```

struct db_getx
{
    int      table_handle; /* table handle returned */
                          /* from open table */
    char     *buffer;      /* pointer to destination buffer */
    int      num_columns;  /* number of columns being */
                          /* selected */
    char     **col_array;  /* array of column names */
                          /* in return order */
    int      num_query_lines; /* number of constraints */
    char     **query_line_array; /* array of query */
                          /* constraints */
    UNSIGNED rec_num;      /* record number (used only */
                          /* by get logical call */
    int      index_handle; /* index handle for */
                          /* TEMP_SORT */
};
typedef struct db_getx db_get;

Utilized by:
    FIRST RECORD
    NEXT RECORD
    PREV RECORD
    LAST RECORD
    FETCH RECORD
    TEMP_SORT

```

Notes

record_number:

DeskMate Technical Reference

Database Resource

Used both on input to the database manager and on output from the database manager. On input, the application indicates the record to fetch after / before. On output, the `record_number` is the database manager number of the record returned to the application.

Example : Application is displaying record number 15, and wishes to display the next record. On input to **NEXT_RECORD** `record_number = 15`. The database manager will locate record number 15 and fetch the next logical record which is record number 5. On output, the `record_number = 5`.

See notes on `db_query` for `query_line_array` format.

The `index_handle` should be either NULL to retrieve records in the defined sort order, or an `index_handle` returned by a **TEMP_SORT** call.

FETCH_RECORD ignores the query information in the structure and returns the requested columns for the requested record. For **FETCH_RECORD** it is required that `num_query_lines` is equal to zero, and `query_line_array` is equal to NULL.

```
struct db_deletex
{
    int     table handle;          /* handle of table to delete */
    UNSIGNED rec num;             /* record number to delete */
    int     num_query_lines;       /* num of constraints */
    char    **query_line_array;    /* array of query */
                                   /* constraints */
};
typedef struct db_deletex db_delete;
```

Utilized by:
DELETE_ROW
DELETE_MULTI_ROWS

Notes

When deleting a single record, the `rec_num` will be used for the delete. The query information will be used to return to the application the next record after the deleted one that matches the query constraints.

When deleting multiple rows, the `rec_num` will be ignored and all rows that match the query constraints will be deleted. If `num_query_lines==0`, all rows in the table will be deleted.

```
struct db_queryx
{
    int     table handle;          /* handle of open table */
    char    *pBuffer;             /* pointer to destination buffer */
    int     num columns;          /* number of cols being selected */
    char    **col_array;          /* array of column names in */
                                   /* return order */
    int     num_query_lines;       /* number of constraints */
    char    **query_line_array;    /* array of query */
                                   /* constraints */
    int     amt_memory;           /* amount of memory application */
                                   /* has for recs */
    int     rec_cnt;              /* number of records in buffer */
    int     cur_rec num;          /* current rec num to be returned */
    int     direction;            /* direction rec are assembled in */
    int     index handle;         /* 0 right now */
    unsigned char q_id;           /* reserved */
};
typedef struct db_queryx db_query;
```


DeskMate Technical Reference Database Resource

table_handle:

The handle returned from an `open_table` request.

num_columns: and col_array:

Allow the user to select a subset of the columns to be returned as records and to specify the order the columns should be arranged in. If `num_columns` equals zero, all columns will be selected and returned in the default order.

num_query_lines: and query_line_array:

Allow the user to define subsets of the data to be returned. Currently the relational operator between queries is AND. The maximum number of queries is twelve. If `num_query_lines` is equal to zero, all the records will be returned.

amt_memory:

Tells the database manager how much data the application can receive. This amount of memory will contain both the data records and an index into those records.

pbuffer:

A pointer to the buffer for the records and index to be returned in which is the size of `amt_memory`.

rec_cnt:

The number of records returned in the buffer.

cur_rec_num:

The number of the last record in the buffer. The data base manager will start after/before this record if `direction` is `DIRN_NEXT` or `DIRN_PRIOR`.

direction:

Informs the database manager of the direction to fetch records in.

/ directions for record searches for multi-user queries */*

`DIRN_FIRST` `FIRST RECORD`

`DIRN_LAST` `LAST RECORD`

`DIRN_NEXT` `NEXT RECORD`

`DIRN_PRIOR` `PREV RECORD`

`DIRN_PRIOR` = Prior records - use `cur_rec_num` to start before.

`DIRN_LAST` = Last records - ignore `cur_rec_num`, start at last record.

`DIRN_NEXT` = Next records - use `cur_rec_num` to start after.

`DIRN_FIRST` = First records - ignore `cur_rec_num`, start at first record.

After using `direction` of `DIRN_FIRST` or `DIRN_LAST`, the user must change the direction to `DIRN_NEXT`, or `DIRN_PRIOR` to get additional records.

index_handle:

Should be NULL to retrieve records in the defined sort order, or an `index_handle` returned by a **TEMP_SORT** call.

q_id:

Reserved for use by the database manager and should not be altered by an application.

Utilized by:

`SETUP QUERY`

`MORE RECORDS`

DeskMate Technical Reference Database Resource

```

struct db_updatex
{
    int    table handle;    /* handle of open table */
    int    n_columns;      /* number of columns being */
                        /* selected */
    char  **col_names;     /* array of column names in */
                        /* return order */
    char  **pOld_col_values; /* old column values (in */
                        /* order) */
    char  **pNew_col_values; /* new column values (in */
                        /* order) */
    UNSIGNED    rec num;    /* current record number */
    char  verify_flag;     /* VERIFY or NO VERIFY */
    UNSIGNED    int    prev_rec; /* previous record, zero means */
                        /* this is first record */
    UNSIGNED    int    next_rec; /* next record, zero means this */
                        /* is last record */
};
typedef struct db_updatex db_update;

```

pOld_col_values:

A pointer to an array that contains the column values as they were before the update. These values will be verified against what is currently contained in the database to verify that the record is current if the verify_flag = VERIFY. If the columns match, then the update can take place.

verify flag:

```

/* defines for verify_flag */
VERIFY      1
NO_VERIFY   0

```

If verify_flag is equal to NO_VERIFY then pOld_col_values **MUST** be NULL.

pNew_col_values:

A pointer to an array that contains the new column values for the update.

Utilized by:

UPDATE_ROW

```

struct handle_bufx    /* general purpose struct for a */
                    /* handle and buffer */
{
    int    handle;    /* handle of open table or file */
    char  *buffer;    /* buffer to return information in */
};
typedef struct handle_bufx handle_buf;

```

Utilized by:

```

GET_TABLE_NAMES
GET_COLUMN_NAMES
GET_MIN
GET_MAX

```

```

struct db_copy_tblx
{
    char  *tablename; /* table name to copy */
    char  data;       /* COPY_DATA or NO_COPY_DATA flag */
};
typedef struct db_copy_tblx db_copy_tbl;

COPY_DATA      1
NO_COPY_DATA   0

```

DeskMate Technical Reference

Database Resource

Utilized by:
db_copy_lay structure

```
struct db_copy_layx
{
    int    file_handle;        /* handle of file containing tables */
                                /* to copy */
    char   *new_filename;      /* name of destination file to create */
    int    num_tables;         /* number of tables in the array */
    db_copy_tbl *table_array;  /* array of tables to copy */
};
typedef struct db_copy_layx db_copy_lay;
```

Utilized by:
COPY_LAYOUTS

Notes

If the data flag in the db_copy_tbl structure is set to COPY_DATA all data in the table will be copied to the new file. If it is set to NO_COPY_DATA only the internal table structure will be copied.

```
struct db_countx
{
    int    table_handle;        /* open table to select from */
    int    num_query_lines;     /* number of constraints */
    char   **query_line_array;  /* array of query constraints */
    int    num_match;           /* number of records that match */
    int    total_num;           /* total number of records */
};
typedef struct db_countx dbcount;
```

Utilized by:
COUNT_RECORDS

Notes

The COUNT_RECORDS call will provide the application with the number of records that match a query and the total number of records in a table. If num_query_lines equals zero then total_num and num_match are equal.

```
struct db_locationx
{
    int    table_handle;        /* handle of open table */
    int    num_query_lines;     /* number of constraints */
    char   **query_line_array;  /* array of query constraints */
    unsigned int    rec_num;     /* rec num to get find */
    unsigned int    prev_num;    /* record number before this one */
    unsigned int    next_num;    /* record number after this one */
    int    index_handle;        /* currently un-used, = 0 */
};
typedef struct db_locationx db_location;
```

Utilized by:
GET_RECORD_LOCATION

DeskMate Technical Reference Database Resource

```
struct      db_indexx
{
    int      table_handle;      /* open table handle */
    char     *index_name;       /* name for this index - can be the */
                                /* same as table name */
    char     *pSortOrder;       /* column names for sort order */
                                /* separated by KS_TKN's */
}
typedef struct db_indexx db_index;
```

Utilized by :

```
    DEFINE_INDEX
    DROP_INDEX
    GET_INDEX_INFO
    TEMP_SORT
```

pSortOrder:

A NULL terminated string that contains column names separated by the KS_TKN character. The last column name in the order must also be terminated with the KS_TKN character, then followed by a NULL terminator.

```
KS_TKN      0x04  /* separator for pSortOrder */
```

```
struct db_end_tempx
{
    int      table_handle;      /* table handle with TEMP_SORT defined */
    int      index_handle;      /* index handle of TEMP_SORT to drop */
};
typedef struct db_end_tempx db_end_temp;
```

Utilized by:

```
    END_TEMP_SORT
```

/* table history slot structure */

```
typedef struct
{
    unsigned int rec_num;       /* record number of transaction */
    char trans_type;            /* A, D, U = Add, Delete, Update */
} hist_slot;
```

/* the following are defines for the transaction type */

```
REC_ADD
REC_DELETE
REC_UPDATE
```

DeskMate Technical Reference

Database Resource

1989 DeskMate 03.03.0x Update Resource

ADD_ROW (pAddStruct)

db_add *pAddStruct;

ADD_ROW adds a row of data to the specified table.

Input Parameters

pAddStruct is a pointer to a db_add structure.

Return Value

Record number of the record just added. A negative value indicates an error has occurred.

Special Notes

Returns the location where the record was added in prev_rec and next_rec elements in the db_add structure.

The previous record number is places in prev_rec. If this value is zero, then the newly added record is the first record. The next record number is places in the next_rec element. If this value is zero, then the newly added record is the last record. If both prev_rec and next_rec are zero, this is the only record in the table.

In a network environment, an entry will be placed in the table history, if the table history has been initialized.

Example

```
erc = db_mgr( ADD_ROW, &AddStruct );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Update Resource

ADD_ROW_NO_FLUSH (pAddStruct)

db_add *pAddStruct;

ADD_ROW_NO_FLUSH adds a row of data to the specified table and does **NOT** force a write to the disk.

Input Parameters

pAddStruct is a pointer to a db_add structure.

Return Value

Record number of the record just added. A negative value indicates an error has occurred.

Special Notes

The location where the record was added is returned via the prev_rec and next_rec elements in the db_add structure.

The previous record number is places in prev_rec. If this value is zero, then the newly added record is the first record. The next record number is places in the next_rec element. If this value is zero, then the newly added record is the last record. If both prev_rec and next_rec are zero, this is the only record in the table.

In a network environment, an entry will be placed in the table history, if the table history has been initialized.

ADD_ROW_NO_FLUSH should only be made in situations where multiple records are being added and will be followed by another database call which will ensure all writes have been forced to the disk. This is **NOT** recommended as a general way to add rows to a table.

An example of an appropriate use of **ADD_ROW_NO_FLUSH** is if the application needs to create a table and initialize it with a certain number of records. Use **ADD_ROW_NO_FLUSH** for all but the last row, and use **ADD_ROW** for the last row. On floppy based systems this will improve performance when initializing a table or file.

Example

```
erc = db_mgr( ADD_ROW_NO_FLUSH, &AddStruct );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Build Resource

ALTER_TABLE (pTableStruct)
db_table *pTableStruct;

ALTER_TABLE enables the user to change the definition of a table. Changes which are supported include: change column name, delete a column, add a column, change a column's type or length.

Input Parameters

pTableStruct is a pointer to a table definition (db_table) structure.

Return Value

<int>

DB_OK if successful.

Possible error codes are:

- DB_TBL_NAME_ERR,
- DB_TABLE_SHARED,
- DB_INVALID_FILE_HANDLE,
- DB_MULTI_UNIQ,
- DB_DUPL_KEY_FIELD.

Special Notes

To alter a table, it must be open for definition modification. If the table is already open, the application must wait to get exclusive use of the table before changing any part of the table definition.

In the db_table structure, the handle will be the handle of the open table.

Only one column can be modified at a time. n_items in the db_table structure **MUST** be set to 1 (one). If the column being modified is a column that defines the sort order, changing its type from N(numeric) to C(character) or K(char88) will change the way the records are ordered. Changing the name of a column does not affect the sort order.

Set unique_col to NULL unless the column is being changed from unique to non-unique or vice versa. To change, set unique_col to SET_UNIQUE_COL or UNSET_UNIQUE_COL. Setting unique column will change the sort order. No other unique column may be specified, or the error DB_MULTI_UNIQ is returned. The field specified as unique must not already contain duplicate data, or the error DB_DUPL_KEY_FIELD is returned, the column will not be changed to unique but the sort order will be changed to the column specified as being a unique column.

Columns that are of type N (numeric) are limited to a maximum size of 39.

Columns should not be changed from type C (character) or K (char88) to N (numeric) if data already exists, since no validation to insure numeric only data takes place.

Example

```
erc = db_mgr( ALTER_TABLE, &TableDefn );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

CLOSE_FILE (FileHandle)

int FileHandle;

CLOSE_FILE notifies the database manager that the user would like to close a particular file on the network or locally. The database manager will verify the file is open and close it.

Input Parameters

FileHandle is the file handle received from the **OPEN_FILE** call.

Return Value

<int>

DB_OK if successful.

DB_INVALID_FILE_HANDLE if the specified file handle is invalid.

Special Notes

If the file still contains open tables, it will not be physically closed, but its use count will be decremented.

When a file is closed, if there are no more users with open tables for that file, all the tables that were opened during the time the file was open will be automatically packed.

When packing a table, the number of deleted rows must be greater than 10% of the number of active rows, for a pack to actually occur.

Example

```
erc = db_mgr( CLOSE_FILE, FileHandle );
```


DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

CLOSE_TABLE (TableHandle)

int TableHandle;

CLOSE_TABLE notifies the database manager that a user has finished using a table. If the table was locked, it can now be unlocked and accessed by other users.

Input Parameters

TableHandle is the table handle which was returned from **OPEN_TABLE**.

Return Value

<int>

DB_OK if successful.

DB_INVALID_TABLE_HANDLE if the specified table handle is invalid.

Special Notes

If a user wants to change the type of access made to a table, for example, to change definition instead of changing data, the user must close the table and re-open it.

Internally, when a table is closed, its use count will be decremented. If the table was previously opened for Definition Modification, the use count will be modified to indicate that table is no longer locked.

Example

```
erc = db_mgr( CLOSE_TABLE, TableHandle );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

COUNT_RECORDS (pCountStruct)

db_count *pCountStruct;

COUNT_RECORDS allows an application to get a count of the number of records that match a specific query.

Input Parameters

pCountStruct is a pointer to db_count structure.

Return Value

<int>

DB_OK if successful.

Possible errors are:

- DB_INVALID_TABLE_HANDLE,
- DB_QUERY_SYNTAX,
- DB_NO_RECORDS,
- DB_NO_ROWS_SELECTED.

Special Notes

COUNT_RECORDS returns two values: the number of records that match, and the total number of records in that table. Every record in the database must be compared to the query, so the time required to complete **COUNT_RECORDS** will vary with the number of records in the database. If the query lines are zero, the total number of records in the file will be returned as the number that match, without reading the records.

Example

```
erc = db_mgr( COUNT_RECORDS, &pCountInfo );
```

DeskMate Technical Reference

Database Resource

1989 DeskMate 03.03.0x Build Resource

COPY_LAYOUTS (pCopyLayStruct)

db_copy_lay *pCopyLayStruct;

COPY_LAYOUTS allows the application to copy selected tables from a database file to a new file (which will be created on the copy). The tables may be copied with or without data.

Input Parameters

pCopyLayStruct is a pointer to a db_copy_lay structure.

Return Value

An integer indicating success or failure.

Special Notes

The new file will be created and each table added to it. For each table added, the corresponding index definition will be copied. If data is to be copied, it will be sorted in the current index order.

NOTE: The database manager does **NOT** check to see if the new filename currently exists. It just creates the new one and would destroy any previous existence of a file with the same name. The database manager does check its list of files currently open for its own use. If the new filename is an existing database file which is open the error DB_ALREADY_OPEN will be returned.

Example

```
erc = db_mgr( COPY_LAYOUTS, &CopyLay );
```

DeskMate Technical Reference

Database Resource

1989 DeskMate 03.03.0x Build Resource

CREATE_FILE (pPathName)

char *pPathName;

CREATE_FILE creates a database file and its structure, and opens the file for use by the database manager.

Input Parameters

pPathName is a pointer to a path name. The pathname **MUST** be a full pathname in upper case including the drive letter.

Return Value

<int>

File handle if the file can be successfully created and opened. Negative value integer error codes if there are problems creating the file.

Possible errors are:

DB_ALREADY_OPEN.

Special Notes

CREATE_FILE will create the file if it doesn't already exist and open it for the database manager use. The database manager does **NOT** check to see if the new filename currently exists. If the new file is an existing database file and it is already open, the error DB_ALREADY_OPEN will be returned. If the file already exists, it will be re-initialized as a database manager file and opened.

Example

```
erc = db_mgr( CREATE_FILE, pPathName );
```

DeskMate Technical Reference

Database Resource

1989 DeskMate 03.03.0x Build Resource

CREATE_TABLE(pTableStruct)
db_table *pTableStruct;

CREATE_TABLE creates a new table in an open and existing database manager file. The new table can be created empty (no columns) or with columns defined at creation time.

Input Parameters

pTableStruct is a pointer to a table definition (db_table) structure.

Return Value

<int>

Table handle if successful.

An error if unsuccessful.

Special Notes

The handle in the db_table structure will be a file handle when creating a new table. When a table is created, a default index will be created that will place records in their physical order.

The table that is created will be opened for Definition Modification. The application can then continue to modify that table's definition by adding columns, deleting columns, changing columns, etc. When an application is ready to add data to the table, the table must be closed and re-opened for data access. It is advisable to define the sort order for the index before closing the table, since defining a sort order is not allowed when a table is open for data access.

The unique_flag in the column structures is ignored. Use **ALTER_TABLE** with the unique_flag set to SET_UNIQUE_COL to specify a unique column.

Columns that are of type N (numeric) are limited to a maximum size of 39.

Example

```
erc = db_mgr( CREATE_TABLE, &TableDefn );
```

DeskMate Technical Reference Database Resource

db_bind_end ()

db_bind_end terminates the bindings to the the database manager resource.

Input Parameters

None.

Return Value

None.

Special Notes

db_bind_end is to be used with **db_bind_init** for 1988 DeskMate 03.02.0x compatability. **db_bind_end** should not be used with **db_bind_read**. **db_bind_end** should be called when no more database support is required by the application, and before **csr_end**.

DeskMate Technical Reference Database Resource

db_bind_init ()

db_bind_init causes the Read, (DBREAD.RES), Update, (DBUPDATE.RES), and Build, (DBBUILD.RES) resources to be loaded into memory.

Input Parameters

None.

Return Value

DB_OK if loading was successful.

DM_ERROR if loading was unsuccessful.

Special Notes

db_bind_init is to be used with **db_bind_end** for 1988 DeskMate 03.02.0x compatability. **db_bind_init** should be called prior to making any database manager requests, and after the **csr_init** call.

Example

```
if ( db_bind_init < DB_OK )  
    /* do not make any database calls */
```

DeskMate Technical Reference

Database Resource

1989 DeskMate 03.03.0x ONLY

db_bind_read (ReadFlag)
int ReadFlag;

db_bind_read allows the application to utilize the 1989 Database Read Resource. **db_bind_read** binds to the 1989 Database Manager Resource DMDB.R89 and the database read resource DBREAD.RES.

Input Parameters

ReadFlag is **READ_ONLY** if the database is to be used for read-only.

ReadFlag is **READ_WRITE** if the database is to be used for updating/building. When specifying **READ_WRITE** the application must call **db_mgr(INIT_DB, BIND_UPDATE)** or **db_mgr(INIT_DB, BIND_BUILD)** depending upon the desired functionality.

Return Value

<int>

DB_OK if successful.

DB_READ_ONLY if only the read-only function is available and the application asked for **READ_WRITE**.

DM_ERROR if the desk executive was unable to find the resource(s) or there was not enough memory to load the resource(s).

Special Notes

db_bind_read should only be used with **db_end_read**.

If the request is made for more than read-only, and the return code is **DB_READ_ONLY**, the application may still continue, however only the read-only functions may be called. The application must still unbind to the resource via **db_end_read**. **DM_ERROR** will cause an error message to appear on the screen. **DB_READ_ONLY** will not cause an error message.

Example

```
/* bind to the 1989 DeskMate 03.03.0x read resource */
if (db_bind_read( READ_WRITE ) == DB_OK)
{
    /* try to bind to the update resource */
    if (db_mgr( INIT_DB, BIND_UPDATE ) == DB_OK)
    {
        .      /* calls to update the database */
        .
        .
        /* unbind to the database update resource */
        db_mgr( END_DB, BIND_UPDATE );
    }
}
/* unbind to the database read resource */
db_end_read();
}
```


DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x ONLY

db_end_read ()

db_end_read terminates the binding to the Database Read Resource. **db_end_read** unbinds to the 1989 Database Manager Resource DMDB.R89 and the database read resource DBREAD.RES.

Input Parameters

None.

Return Value

None.

Special Notes

db_end_read should only be used with **db_bind_read**. **db_end_read** should be called when no more database read support is required by the application.

See Also

db_bind_read()
INIT_DB

Example

```
/* bind to the 1989 DeskMate 03.03.0x read resource */
if (db_bind_read( READ_WRITE ) == DB_OK)
{
    /* try to bind to the update resource */
    if (db_mgr( INIT_DB, BIND_UPDATE ) == DB_OK)
    {
        .      /* calls to update the database */
        .
        /* unbind to the database update resource */
        db_mgr( END_DB, BIND_UPDATE );
    }
    /* unbind to the database read resource */
    db_end_read();
}
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Build Resource

DEFINE_INDEX (pIndexStruct)
db_index *pIndexStruct;

DEFINE_INDEX defines the sort order for the disk resident index.

Input Parameters

pIndexStruct is a pointer to a db_index structure.

Return value

<int>

An integer indicating success or failure.

Possible errors:

DB_UNIQ_NO_SORT.

Special Notes

An index definition is a string concatenated column names separated by a KS_TKN delimiter. The last column name in the sort order string **MUST ALSO** be terminated by a KS_TKN, then followed by the string-terminating NULL. To define an index, the table must be opened for definition modification. Currently only one (1) index per table is supported, but an index name is provided for future support of multiple indexes. When the index is defined, any previous index will be deleted and a new index created by reading and sorting all the records in that table. If a unique column is specified, it must be the first column in the sort order or the error DB_UNIQ_NO_SORT is returned.

Example

```
erc = db_mgr( DEFINE_INDEX, &Index );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Update Resource

DELETE_MULTI_ROWS (pDeleteStruct)
db_delete *pDeleteStruct;

DELETE_MULTI_ROWS deletes all rows or a specified subset of rows in a given table.

Input Parameters

pDeleteStruct is a pointer to a db_delete structure.

Return Value

<int>

The number of rows deleted if successful.

A negative error code will be returned if unsuccessful .

Special Notes

If num_query_lines is zero, all rows in the table will be deleted, otherwise only those records that match the query constraints will be deleted. The rec_num will be ignored.

In a network environment a table history will be updated to indicate multiple records in a table have been deleted, if table history has been initialized.

Example

```
erc = db_mgr( DELETE_MULTI_ROWS, pDelete );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Update Resource

DELETE_ROW (pDeleteStruct)
db_delete *pDeleteStruct;

DELETE_ROW deletes a specific row from a table.

Input Parameters

pDeleteStruct is a pointer to a db_delete structure.

Return Value

<int>

The next record number.

Possible error codes are:

- DB_RECNUM_ERR,
- DB_OK,
- DB_NO_ROWS_SELECTED.

Special Notes

If a record with the specified rec_num doesn't exist, DB_RECNUM_ERR will be returned. The application will determine if deleting a non-existent record is an error. When a record is deleted the next matching record number will be returned. If the last record was deleted, the new last record number will be returned. If there are no more records in the file after the delete DB_OK will be returned. The query lines are used only to determine the next matching record. If no more records match the query but there are still records in the file, the error DB_NO_ROWS_SELECTED will be returned. If num_query_lines is equal to zero then all records match and the next record number will be returned.

In a network environment, an entry for the delete will be placed in the table history, if table history has been initialized.

Example

```
erc = db_mgr( DELETE_ROW, &DeleteStruct );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Build Resource

DROP_INDEX (pIndexStruct)

db_index *pIndex;

DROP_INDEX removes an index from a table so that the records are presented in physical order.

Input Parameters

pIndex is a pointer to db_index structure.

Return Value

<int>

An integer indicating success or failure.

Special Notes

The table must be open for definition modification.

Currently, because only one index per table is allowed, the table_handle is required in the db_index structure.

index_name should be either the actual name or NULL, pSortOrder must be NULL.

Example

```
erc = db_mgr( DROP_INDEX, &Index );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Build Resource

DROP_TABLE (pTableStruct)

table_access *pTableStruct;

DROP_TABLE deletes a table and all its data from the database file.

Input Parameters

pTableStruct is a pointer to a table structure.

Return Value

<int>

DB_OK if successful.

Possible errors are:

- DB_TBL_NAME_ERR,
- DB_TABLE_OPEN,
- DB_TABLE_SHARED,
- DB_INVALID_FILE_HANDLE.

Special Notes

The file must be open. The table to be deleted must be closed. Space occupied by that table will be marked as free for other uses by the database manager.

Example

```
erc = db_mgr( DROP_TABLE, &pTableStruct );
```

DeskMate Technical Reference

Database Resource

1989 DeskMate 03.03.0x ONLY

END_DB (Resource)

int Resource;

END_DB will unbind database resources from the Database Resource Manager.

Input Parameters

Resource if BIND_UPDATE the application may no longer access the database update functions.

Resource if BIND_BUILD the application may no longer access the database build functions.

Resource if BIND_ALL the application may no longer access any of the database functions.

Return Value

None.

Special Notes

END_DB must be called before db_end_read.

Example

```
/* unbind to the build resource */  
erc = db_mgr ( END_DB, BIND_BUILD )
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Update Resource

END_TEMP_SORT (pEndSortStruct)

db_end_sort *pEndSortStruct;

END_TEMP_SORT informs the database manager that an application is finished with a temporary sort and it may be deleted.

Input Parameters

pEndSortStruct is a pointer to a db_end_sort structure.

Return Value

<int>

DB_OK if successful.

Possible error codes are:

DB_INVALID_TABLE_HANDLE,
DB_INVALID_INDEX_HANDLE.

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

FETCH_RECORD (pGetStruct)

db_get *pGetStruct;

FETCH_RECORD retrieves a specific record by record number. The num_columns and col_array must be specified.

Input Parameters

pGetStruct is a pointer to a db_get structure.

Return Value

<int>

DB_OK if successful.

Possible error codes are:

- DB_INVALID_TABLE_HANDLE,
- DB_LAST_RECORD,
- DB_FIRST_RECORD,
- DB_ONLY_RECORD,
- DB_DISK_READ,
- DB_RECNUM_ERR.

Special Notes

The user sends a record number to the database manager. The database manager will fetch that record and return it to the user. The query information in the db_get structure will be ignored. However, num_query_lines should be set to NULL and query_line_array should be set to NULL; these are necessary for the bindings.

DB_ONLY_RECORD is returned if there's only 1 record in the table.

The receiving buffer must be large enough for each column requested plus a NULL terminator for each column PLUS a 1-byte end of record token (ER_TKN).

The index handle in the db_get structure should be either NULL, to retrieve records in the defined sort order, or an index_handle returned by a **TEMP_SORT** call.

Example

```
erc = db_mgr( FETCH_RECORD, & db_get );
```

This function has limited usefulness, since there is no rhyme or reason to the record numbering. I.e., record #1 may not exist, and if it does, may not be the first record.

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

FIRST_RECORD (pGetStruct)

db_get *pGetStruct;

FIRST_RECORD returns a single record to the user's buffer. The record number in the db_get structure will be utilized on output. The database manager will return the record number of the first record.

Input Parameters

pGetStruct is a pointer to a db_get structure.

Return Value

<int>

DB_OK if successful.

Possible error codes are:

- DB_LAST_RECORD,
- DB_INVALID_TABLE_HANDLE,
- DB_DISK_READ,
- DB_NO_ROWS_SELECTED,
- DB_NO_RECORDS,
- DB_RECNUM_ERR.

Special Notes

FIRST_RECORD returns a single record to the user's buffer. It will contain the requested columns in the requested order. The query constraints will be upheld with each record returned. The record number is vital to maintaining the correct order. The database manager will utilize the on-disk index to scan through the table.

Explanation of errors returned:

DB_NO_ROWS_SELECTED there are records in table, but none match.

DB_NO_RECORDS the table has NO records.

DB_RECNUM_ERR the record number in db_get is invalid.

DB_LAST_RECORD the last record is returned from a **NEXT_RECORD** or **FIRST_RECORD** call.

The buffer the user receives the data in must be large enough for each column requested plus a NULL terminator for each column PLUS a 1 byte ER_TKN.

The index handle in the db_get structure should be either NULL, to retrieve records in the defined sort order, or an index_handle returned by a **TEMP_SORT** call.

Example

```
erc = db_mgr( FIRST_RECORD, & db_get );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

GET_COL_INFO (pTableStruct)
db_table *pTableStruct;

GET_COL_INFO fills in the column information for a given array of column names.

Input Parameters

pTableStruct is a pointer to a db_table structure with the table_handle field initialized to an open table handle, the name pointing to a NULL byte, the update_type set to COLUMN_INFO, and the number of items set to the number of columns you are retrieving information for. Each of the column structures should have the column name filled in and the new name set to NULL.

Return Value

DB_OK if no errors.

DB_COL_NAME_ERR if any column not found in given table .

DB_INVALID_TABLE_HANDLE if table is not open.

Special Notes

The application should call **GET_COL_NAMES** prior to **GET_COL_INFO** to ensure the column names are filled in for each db_column structure passed. When zero (0) is specified for the number of columns, no column information is returned.

Example

```
if( erc = db_mgr( GET COL INFO, &pTbl ) )  
    print_err( erc );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

GET_COLUMN_NAMES (pHandleBufStruct)

handle_buf *pHandleBufStruct;

GET_COLUMN_NAMES retrieves the names of the columns in a table.

Input Parameters

pHandleBufStruct is a pointer to a **handle_buf** structure. The handle **MUST** be a table handle.

Return Value

<int>

The number of columns.

A negative number will indicate that the operation was unsuccessful.

Special Notes

The table must be open. The column names will be returned in the buffer separated by NULL characters. There will be an ER_TKN after the final NULL.

Example

```
erc = db_mgr( GET_COLUMN_NAMES, pInfo );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

GET_INDEX_INFO (pIndexStruct)

index_struct *pIndexStruct;

GET_INDEX_INFO retrieves index information about a particular table. The column names that make up the sort order will be returned.

Input Parameters

pIndexStruct is a pointer to **index_struct**.

Return Value

<int>

The number of fields in the sort order. A negative number for failure.

Possible error values are:

DB_INVALID_TABLE_HANDLE,
DB_COL_NAME_ERR.

Special Notes

The sort columns names are returned to the application in a single string with the column names separated by a KS_TKN delimiter. The file and table associated with the index must be already open. The table handle is required in the **db_index** structure, the **index_name** must be a pointer to a valid name or NULL. **pSortOrder** must point to a buffer that is big enough for the column names and KS_TKN's. There will also be a terminating NULL and an ER_TKN.

If there are NO defined sort field the buffer will contain the NULL and ER_TKN.

Example

```
erc = db_mgr( GET_INDEX_INFO, &IndexInfo );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

GET_MAX (pHandleBufStruct)

handle_buf *pHandleBufStruct;

GET_MAX returns the highest value for the given column name and the record number of the first record with that value. To access this record, use **FETCH_RECORD** with the returned record number.

Input Parameters

pHandleBufStruct is a pointer to a **handle_buf** structure. The handle must be an open table handle. The buffer should contain the name of a valid column in the open table.

Return Value

<int>

The record number if successful.

Possible error codes:

- DB_INVALID_TABLE_HANDLE,
- DB_COL_NAME_ERR,
- DB_NO_RECORDS,
- DB_DISK_READ.

Special Notes

The buffer will contain on output a string of the maximum value terminated by a NULL and an ER_TKN.

Example

```
rec_num = db_mgr ( GET_MAX, &pInfo );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

GET_MIN (pHandleBufStruct)
handle_buf *pHandleBufStruct;

GET_MIN returns the lowest value for the given column name and the record number of the first record with that value. To access this record, use **FETCH_RECORD** with the returned record number.

Input Parameters

pHandleBufStruct is a pointer to a **handle_buf** structure. The handle must be an open table handle. The buffer should contain the name of a valid column in the open table.

Return Value

<int>

The record number if successful.

Possible error codes:

DB_INVALID_TABLE_HANDLE,
DB_COL_NAME_ERR,
DB_NO_RECORDS,
DB_DISK_READ.

Special Notes

The buffer will contain on output a string of the minimum value terminated by a NULL, then the ER_TKN.

Example

```
rec_num = db_mgr ( GET_MIN, &pInfo );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

GET_PAGE_SETUP (pHandleBufStruct)

handle_buf *pHandleBufStruct;

GET_PAGE_SETUP gets the page setup information that is stored in the file.

Input Parameters

pHandleBufStruct is a pointer to a **handle_buf** structure. The handle **MUST** be a file handle.

Return Value

An integer indicating success or failure.

Special Notes

The application should place that address of their page setup structure in the buffer pointer provided by the **handle_buf** structure. Type casting should be used to prevent compiler warning. For example:

```
pInfo->buffer = (char *)&page_setup_info;
```

Example

```
db_mgr( GET_PAGE_SETUP, &pInfo );
```


DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

GET_RECORD_LOCATION (pLocationStruct)

db_location *pLocationStruct;

GET_RECORD_LOCATION is given a table handle, record number, and any desired query constraints. The database manager will fill in the matching previous and next record numbers.

Input Parameters

pLocationStruct is a pointer to a **db_location** structure. **index_handle** must be equal to zero.

Return Value

<int>

DB_OK if successful.

An error if unsuccessful.

Special Notes

GET_RECORD_LOCATION is especially useful in a network environment for an application that is making use of the multi-query calls, and table history. When table history indicated a new record has been added, the application can determine it's location and if it matches the current query constraints in effect to know if the application needs to fetch this new record.

If the record being located is the first one in the file, or the first one to match the query constraints, zero will be returned as the **prev_num**.

If the record being located is the last one in the file, or the last one to match the query constraints, zero will be returned as the **next_num**.

The **index_handle** is currently ignored, and record locations are returned based on the permanent sort order only.

Example

```
erc = db_mgr ( GET_RECORD_LOCATION, &location );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

GET_TABLE_NAMES (pHandleBufStruct)

handle_buf *pHandleBufStruct;

GET_TABLE_NAMES returns the names of the tables which are in the file. The table names will be in the buffer separated by NULL characters.

Input Parameters

pHandleBufStruct is a pointer to a **handle_buf** structure. The handle **MUST** be a file handle.

Return Value

<int>

The number of tables in the file.

If unsuccessful, an error will be returned.

Special Notes

The table names stored in the directory for the file are retrieved and returned to the user. The file must be open. The table names are separated by NULL characters in the buffer. There will be an ER_TKN after the final NULL.

DBCOLS an internal database manager table will be included and returned with this request. It is not allowed for a user application to open DBCOLS.

Example

```
num_tables = db_mgr( GET_TABLE_DIR, &pInfo );
```

DeskMate Technical Reference

Database Resource

1989 DeskMate 03.03.0x ONLY

INIT_DB (Resource)

int Resource;

INIT_DB will bind database resources to the Database Resource Manager.

Input Parameters

Resource if BIND_UPDATE the application may access the database update functions.

Resource if BIND_BUILD the application may access the database build functions.

Resource if BIND_ALL the application may access all of the database functions.

Return Value

DB_OK if successful.

DB_OUT_OF_MEMORY if the resource requested could not be loaded.

DB_READ_ONLY if an attempt was made to load a resource and **db_bind_read** bindings returned a DB_READ_ONLY.

Special Notes

db_bind_read must return DB_OK before calling **INIT_DB**. A return of DB_OUT_OF_MEMORY will cause an error message to appear to the user. DB_READ_ONLY will not display an error message.

Example

```
/* call db bind read */
/* see db Bind Read */
/* bind to the build resource */
erc = db_mgr ( INIT_DB, BIND_BUILD )
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

LAST_RECORD (pGetStruct)

db_get *pGetStruct;

LAST_RECORD allows the application to fetch a single record at a time and returns the record to the user's buffer. The record number in the db_get structure will be utilized both on input and output. The database manager will return the record number of the last record.

Input Parameters

pGetStruct is a pointer to a db_get structure.

Return Value

<int>

DB_OK if successful.

Possible error codes are:

- DB_FIRST_RECORD,
- DB_INVALID_TABLE_HANDLE,
- DB_DISK_READ,
- DB_NO_ROWS_SELECTED,
- DB_NO_RECORDS,
- DB_RECNUM_ERR.

Special Notes

LAST_RECORD returns a single record to the user's buffer. The buffer will contain the requested columns in the requested order. The query constraints will be up-held with each record returned. The record number is vital to maintaining the correct order. The database manager will utilize the on-disk index to scan through the table.

Explanation of errors returned:

DB_NO_ROWS_SELECTED there are records in table, but none match.

DB_NO_RECORDS the table has NO records.

DB_RECNUM_ERR the record number in db_get is invalid.

DB_FIRST_RECORD the first record is returned from a **PREV_RECORD** or **LAST_RECORD** call.

The buffer the user receives the data in must be large enough for each column requested plus a NULL terminator for each column PLUS a 1 byte ER_TKN.

The index handle in the db_get structure should be either NULL, to retrieve records in the defined sort order, or an index_handle returned by a **TEMP_SORT** call.

Example

```
erc = db_mgr( LAST_RECORD, & db_get );
```

DeskMate Technical Reference Database Resource

Support for the Merge function deleted.

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

MORE_RECORDS (pQueryStruct)

db_query *pQueryStruct;

MORE_RECORDS requests more records that match the query constraints. The user specifies the beginning record number and the direction in which to search for more records (first, next, prior, last). The index_handle should be either NULL to retrieve records in the defined sort order, or an index_handle returned by a **TEMP_SORT** call.

Input Parameters

pQueryStruct is a pointer to a db_query structure.

Return Value

<int>

DB_OK if successful.

Possible errors codes are:

- DB_INVALID_TABLE_HANDLE,
- DB_QUERY_SYNTAX,
- DB_NO_ROWS_SELECTED,
- DB_LAST_RECORD,
- DB_FIRST_RECORD,
- DB_NO_RECORDS,
- DB_DISK_READ.

Special Notes

If the user sends a direction of first or last, then the record number will be ignored and the first (or last) x records that will fit in the user's memory area will be returned. First arranges records in Ascending order. Last arranges records in Descending order.

The current record number will be returned automatically upon returning from **MORE_RECORDS** so subsequent calls to **MORE_RECORDS** can normally be made without specifying a number.

The user **CANNOT** change query constraints or requested columns between calls: **SETUP_QUERY**, **MORE_RECORDS** and subsequent calls to **MORE_RECORDS**.

Errors returned:

DB_NO_ROWS_SELECTED if there are records in the table, but none match the query.

DB_NO_RECORDS if there are no physical records in the table.

DB_FIRST_RECORD, DB_LAST_RECORD these are returned when the FIRST or LAST record is being returned. If requesting records in FIRST or NEXT order, and both the FIRST and LAST record is in the buffer only DB_LAST_RECORD will be returned. If requesting records in the LAST or PREV order and both the FIRST and LAST record is in the buffer, only DB_FIRST_RECORD will be returned.

If only 1 record matches the query, rec_cnt will be set to 1, and either DB_LAST_RECORD or DB_FIRST_RECORD will be returned. This is how an application will determine that there is only 1 record matching. rec_cnt can also be equal to 1 in the case where only 1 record will fit in the buffer, in this case DB_OK will be the return code.

DeskMate Technical Reference Database Resource

Example

```
erc = db_mgr( MORE_RECORDS, &db_query );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

NEXT_RECORD (pGetStruct)

db_get *pGetStruct;

NEXT_RECORD fetches a single record at a time and returns it to the user's buffer. The record number in the db_get structure will be utilized both on input and output. The application will indicate its current position in the table via the record number and the database manager will return the record number of the requested record.

Input Parameters

pGetStruct is a pointer to a db_get structure.

Return Value

<int>

DB_OK if successful.

Possible error codes are:

- DB_LAST_RECORD,
- DB_INVALID_TABLE_HANDLE,
- DB_DISK_READ,
- DB_NO_ROWS_SELECTED,
- DB_NO_RECORDS,
- DB_RECNUM_ERR.

Special Notes

NEXT_RECORD returns a single record to the user's buffer. The buffer will contain the requested columns in the requested order. The query constraints will be up-held with each record returned. The record number is vital to maintaining the correct order. The database manager will utilize the record number and on-disk index to scan through the table.

Explanation of errors returned:

DB_NO_ROWS_SELECTED there are records in table, but none match.

DB_NO_RECORDS the table has NO records.

DB_RECNUM_ERR the record number in db_get is invalid.

DB_LAST_RECORD the last record is returned from a **NEXT_RECORD** or **FIRST_RECORD** call.

The buffer the user receives the data in must be large enough for each column requested plus a NULL terminator for each column PLUS a 1 byte ER_TKN.

The index handle in the db_get structure should be either NULL, to retrieve records in the defined sort order, or an index_handle returned by a **TEMP_SORT** call.

Example

```
erc = db_mgr( NEXT_RECORD, & db_get );
```


DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

OPEN_FILE (pPathName)

char *pPathName;

OPEN_FILE notifies the database manager that a user wants particular local or network file. The database manager verifies the file is of the proper type and opens it.

Input Parameters

pPathName is a pointer to a full network pathname. If the pathname does not contain network routing information, it will be assumed to reside on the current machine. The pathname **MUST** be a full pathname in upper case including the drive letter.

Return Value

The file handle if the file is successfully opened.

If the file is not successfully opened, an error value will be returned.

Special Notes

When a file is opened, the database manager checks to see if the file has another user. If so, the same file handle will be returned. A use count is kept for each open file to know when it can be physically closed.

Example

```
FileHandle = db_mgr( OPEN_FILE, pPathName );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

OPEN_TABLE (pTableAccessStruct)

table_access *pTableAccessStruct;

OPEN_TABLE notifies the database manager of the table name within a file that a user wishes to utilize. This will allow the database manager to set up in memory information about the table and keep a count of how many users are accessing that table.

Input Parameters

pTableAccessStruct is a pointer to a pointer to a table structure.

Return Value

<int>

The table handle which will be required in future query and record fetches for this table.

All error codes will be negative numbers to prevent conflict with the table handle.

Possible error conditions are:

DB_TABLE_LOCKED
DB_TBL_NAME_ERR.

Special Notes

OPEN_TABLE must be made after the user has opened a Database file with either **OPEN_FILE** or **CREATE_FILE**. **OPEN_TABLE** will bring into the database manager's memory information about the columns in that table. When a user is accessing a table for Definition Modification only that user will be allowed access to that table. If other users attempt to open a table that is locked due to Definition Modification, those users will receive an error indicating that table is locked. When a user has completed the Definition of a table, it is very important that the user closes the table and re-open it with a level of Data Access/Modification.

During the time that a user has a table open for Definition Modification, the user will be able to perform the following operations in addition to those operations available under Data Access:

Add column to table.

Delete a column from a table.

Change the definition of a column (length, type, name).

Pack a table.

Change the sort order (involves rebuilding the index).

When a user has a table open for Data Access the following operations may be performed:

Add row of data (multi-user or single user).

Delete a row of data (multi-user or single user).

Update a row of data (multi-user or single user).

Place query constraints on data to be returned to the application.

Fetch records (1 or more at a time).

Fetch specific record by record number.

Retrieve table definition information - column in the table.

Retrieve information about the file - tables in the file.

Retrieve information about the index for a table.

Delete all rows of data from the table (or a subset of rows).

Internally, when a table is opened it's use_count will be incremented. When a table is open for "Definition Modification" the use count will be marked with a special value to indicate that it is locked.

DeskMate Technical Reference Database Resource

Example

```
erc = db_mgr( OPEN_TABLE, &pTableStruct );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Build Resource

PACK_TABLE (TableHandle)

int TableHandle;

PACK_TABLE physically deletes records that have been marked as deleted in the database.

Input Parameters

TableHandle is a handle of a table open for definition modification.

Return Value

<int>

An integer indicating success or failure.

Example

```
erc = db_mgr( PACK_TABLE, tbl_handle );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

PREV_RECORD (pGetStruct)

db_get *pGetStruct;

PREV_RECORD fetches a single record at a time. The record number in the db_get structure will be utilized both on input and output. During input, the application uses this field to indicate its current position in the database; during output, the database manager uses this field to return the record number of the requested record.

Input Parameters

pGetStruct is a pointer to a db_get structure.

Return Value

<int>

DB_OK if successful.

Possible error codes are:

- DB_FIRST_RECORD,
- DB_INVALID_TABLE_HANDLE,
- DB_DISK_READ,
- DB_NO_ROWS_SELECTED,
- DB_NO_RECORDS,
- DB_RECNUM_ERR.

Special Notes

PREV_RECORD returns a single record to the user's buffer. It will contain the requested columns in the requested order. The query constraints will be up-held with each record returned. The record number is vital to maintaining the correct order. The database manager will utilize the record number and on-disk index to scan through the table.

Explanation of errors returned:

DB_NO_ROWS_SELECTED there are records in table, but none match.

DB_NO_RECORDS the table has NO records.

DB_RECNUM_ERR the record number in db_get is invalid.

DB_FIRST_RECORD the first record is returned from a **PREV_RECORD** or **LAST_RECORD** call.

The buffer the user receives the data in must be large enough for each column requested plus a NULL terminator for each column PLUS a 1 byte ER_TKN.

The index handle in the db_get structure should be either NULL, to retrieve records in the defined sort order, or an index_handle returned by a **TEMP_SORT** call.

Example

```
erc = db_mgr( PREV_RECORD, & db_get );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Update Resource

SAVE_PAGE_SETUP (pHandleBufStruct)

handle_buf *pHandleBufStruct;

SAVE_PAGE_SETUP stores the page setup information in the file.

Input Parameters

pHandleBufStruct is a pointer to a **handle_buf** structure. The handle **MUST** be a file handle.

Return Value

An integer indicating success or failure.

Special Notes

The application should place the address of their page setup structure in the buffer pointer provided by the **handle_buf** structure. Type casting should be used to prevent compiler warning. For example:

```
pInfo->buffer = (char *)&page_setup_info;
```

Example

```
db_mgr( SAVE_PAGE_SETUP, &pInfo );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Read Resource

SETUP_QUERY (pQueryStruct)
db_query *pQueryStruct;

SETUP_QUERY informs the database manager of the query constraints that the user would like to place upon the records returned. The user may specify up to MQLINES query constraints each being related with the AND operator. The user may also indicate the columns to be returned and their respective order in a record. **SETUP_QUERY** will setup the query info in the database manager and the first set of matching records will be returned to the user.

Input Parameters

pQueryStruct is a pointer to a db_query structure.

Return Value

<int>

DB_OK if successful.

Possible errors are:

DB_INVALID_TABLE_HANDLE,
DB_QUERY_SYNTAX,
DB_NO_ROWS_SELECTED,
DB_LAST_RECORD,
DB_FIRST_RECORD,
DB_NO_RECORDS.

Special Notes

See Notes under **MORE_RECORDS** for description of errors.

All records that match the query will be returned in the sort order of the specified index. The index handle must be set to NULL to retrieve records in the default sort order, or to the index handle returned by a **TEMP_SORT** call. There is a provision in the db_query structure to allow for multiple indexes in the future.

The format of a query is as follows:

column_name <operator> null terminated value.

column_name must be a valid column name for that table.

<operator> must be one of the following

DB_EQUAL	'='	all records equal to the specified value.
DB_GREATER	'>'	all records greater than or equal to the specified value.
DB_LESS	'<'	all records less than or equal to the specified value.
DB_NOT_EQUAL	'!'	all records NOT equal to the specified value.
DB_SUBSTRING	' '	all records with this substring in the specified field.
REC_CONTAINS	'^'	all records which contain the given text string in any column.

null terminated value can contain the following wild cards :

- * - match all characters at and after this character position.
- ? - match this character with any character.

DeskMate Technical Reference Database Resource

Note: When the operator is DB_SUBSTRING, a '?' will match any character in that position. An '*' will match only another '*' in the same position. It **WILL NOT** act as a wildcard in DB_SUBSTRING only.

All records that are returned in the user's buffer are terminated with an end of record token (ER_TKN).

There should NOT be any extraneous white space in the query. The database manager will scan for the operator and if one of the above is NOT present, an error will be returned. Example : "Last Name=Cross"

Since the column Last Name contains an intermediate space, it must be included to match the column name.

Along with the data records, an index into those data records will be returned by the database manager. It will contain a 2 byte offset into the buffer for each record location and a 2 byte corresponding record number.

The user will have only 1 memory area in which to receive both the data records and an index into those records. When the user has requested records in the "next" direction, the records will start at the beginning of memory and grow towards the end. The index will start at the end and grow towards the beginning. When the two meet or would overlap, that determines how much data can be returned to the user. This would look something like:

buf[0]

Data record 1
Data record 2
Data record 3
.....
.....
Data record n
offset n rec num n off- set 3 rec num 3 offset 2 rec num 2 offset 1 rec num 1

buf[nnnnn]

If the user has requested records in the "previous" direction, the data records will be place in the buffer in reverse order, and the index will be built in reverse order. The direction flag in the structure will be the key to keeping track of the record order. If the application requests the last record and there is enough data area for 5 records, they would be arranged as:

buf[0]

Last Data record n
Data record n - 1
Data record n - 2
.....
Data record n - 4
offset n-4 rec num n-4 off- set n-2 recnum n-2 offset n-1 recnum n-1 offset n recnum n

buf[nnnnn]

DeskMate Technical Reference Database Resource

Assuming that RelativeRecordNum begins with zero and pbufend points to the byte past the end of the buffer, the user can then access the index with the following formulas;

```
RecordNum = (unsigned short*) ( pbufend - ( RelativeRecNum  
                                * INDEX_OVERHEAD + 2 ) )  
RecordOffset = (unsigned short*) ( pbufend - ( RelativeRecNum  
                                * INDEX_OVERHEAD + 4 ) )  
INDEX_OVERHEAD = 4.
```

Example

```
erc = db_mgr( SETUP_QUERY, &pquery );
```

See Also

MORE_RECORDS()

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Update Resource

TEMP_SORT (pIndexStruct)
db_index *pIndexStruct;

TEMP_SORT defines a temporary sort order for the current records in a table. This will not affect the permanent sort that is defined by **DEFINE_INDEX**. Any record additions, deletions, or updates will not be reflected in the **TEMP_SORT** after the **TEMP_SORT** has been defined.

Input Parameters

pIndexStruct is a pointer to a db_index structure.

Return Value

<int>

The index handle if successful.

Special Notes

If the application is running in a multi-user environment, and a record is added, deleted or updated in the main index, that action will not be reflected in the temporary index. This can cause the error **DB_RECORD_DELETED** to be returned for a **FETCH_RECORD** call, if the record being fetched was deleted **AFTER** the **TEMP_SORT** was called. If a deleted record is encountered when processing **FIRST**, **NEXT**, **PREV**, **LAST** the database manager will skip that deleted record and continue with the **NEXT**/**PREV** one.

ONLY ONE TEMP_SORT may be in effect for a table at one time. If a **TEMP_SORT** is already in effect, an error of **DB_TEMP_SORT_ALREADY** will be returned.

Example

```
index_handle = db_mgr ( TEMP_SORT, pIndex );
```

DeskMate Technical Reference Database Resource

1989 DeskMate 03.03.0x Update Resource

UPDATE_ROW (pUpdateStruct)

db_update *pUpdateStruct;

UPDATE_ROW updates the row specified in the structure.

Input Parameters

pUpdateStruct is a pointer to a db_update structure

Return Value

<int>

DB_OK if successful.

An error if unsuccessful.

Special Notes

There is a verify_flag in the structure that should be set to TRUE in a multi-user environment. If the verify_flag is equal to TRUE, the old_col_values will be compared against what is on disk and perform the update only if the record is current. If the record is not current, DB_RECORD_NOT_CURRENT will be returned.

If verify_flag is equal to FALSE, pOld_col_values **MUST** be set to NULL.

The location where the record was updated is returned via the prev_rec and next_rec elements in the db_update structure.

The previous record number is places in prev_rec. If this value is zero, then the newly updated record is the first record. The next record number is places in the next_rec element. If this value is zero, then the newly updated record is the last record. If both prev_rec and next_rec are zero, this is the only record in the table.

In a network environment, an entry will be placed in the table history.

Example

```
erc = db_mgr( UPDATE_ROW, &AddStruct );
```

DeskMate Technical Reference Database Resource

Explanation of Error Codes returned by the Data Base Manager

Code	Explanation
DB_ALREADY_OPEN.....	An attempt to create a database did not succeed because the named database is already opened.
DB_BUSY.....	The database manager is busy. A client on a server machine can get this error. Try again after a delay. If the call to beclient had TRUE for the box_flag parameter this is returned when the user CANCEL's from a busy RETRY/CANCEL message box.
DB_COL_NAME_ERR.....	The column name in a structure does not match any of the column names for the given table and file handles.
DB_DIFFERENT_TABLES.....	The tables given to merge have differing structures in column number, type, or length, and so cannot be merged.
DB_DIR_FULL.....	Index or table could not be created because there is no directory space left in the database. An existing table or index will have to be deleted to make more room.
DB_DISK_FULL.....	The physical disk is full, or the file has reached its maximum size. Last request was not completed.
DB_DISK_READ.....	Low-level file I/O error. Physical read from disk returned error code.
DB_DISK_WRITE.....	Low-level file I/O error. Physical write to disk returned error code.
DB_DUPL_COLUMN.....	The column name given for a column to be added is the same as an already existing column in the same table and file. Column names within a table must be unique.
DB_DUPL_KEY_FIELD.....	An attempt was made to add or update a record which resulted in non-unique data for a column specified as unique (the add/update did not occur). Also returned when a column newly specified as unique contains duplicate values such as in the ALTER_TABLE call when an attempt was made to change a column type to unique, but data already exists for that column which is not unique. The sort order gets changed to that column and the records are resorted, but unique is not maintained.
DB_DUPL_TABLE.....	Duplicate table name. A table to be added has the same name as an already existing table. Table names must be unique within a file.
DB_FIRST_RECORD.....	Condition code. The first record was found during a query or fetch. This code is returned when the first record is in the user's record buffer. This is returned for the calls FETCH_RECORD , LAST_RECORD or PREV_RECORD , or the direction in the query structure is DIRN_LAST or DIRN_PRIOR .
DB_INVALID_CHAR.....	A non-ASCII character that is not a token was found in a database record to be added. Only ascii characters and tokens are allowed as database data.
DB_INVALID_FILE_HANDLE....	The file handle given does not match any open file.
DB_INVALID_FILE_TYPE.....	An attempt was made to open a non-database file.
DB_INVALID_TABLE_HANDLE...	The table handle given is not valid

DeskMate Technical Reference

Database Resource

DB_LAST_RECORD.....Condition code. The last record was found during a query or fetch. This code is returned when the last record is in the user's record buffer returned by the database and either the call using the get structure is **FIRST_RECORD** or **NEXT_RECORD**, or the direction in the query structure is **DIRN_FIRST** or **DIRN_NEXT**. It is also returned when a record fetched by record number is the last record.

DB_MAX_NBR_FIELDS.....An attempt to add a column failed because the table has the maximum number of columns allowed already.

DB_MULTI_AGAIN.....Internal.

DB_MULTI_UNIQ.....An attempt was made to specify another unique column when one already exists.

DB_NETERR.....Network error.

DB_NO_CREATE.....Low-level file I/O error. Error code returned on create file call. The file could not be created.

DB_NO_DBCOLUMNS.....The directory entry for DB_COL was not found so no columns could be added. Your database file is corrupted and cannot be used. Fatal error.

DB_NO_EOR.....A database record to be added has no end-of-record token. An end-of- record token is required to distinguish valid record data from left-over random buffer trash. This can also happen if a record is bigger than MAX_REC_SIZE.

DB_NO_INDEX.....No index exists for a file being queried. This error should not occur, since a default physical-order index is maintained for files with no index. Also returned when NO_DUP is specified on a merge but the destination table has no sort order.

DB_NO_OPEN.....Low-level file I/O error. Error code returned an open file call. The file could not be opened.

DB_NO_PACK.....Condition code. The number of deleted rows in a table should equal at least 10% of the total number of rows to justify the packing overhead. This error means no packing was done.

DB_NO_RECORDS.....Record(s) were requested from a table with no records.

DB_NO_ROWS_SELECTED.....On a valid query to a database with records, no records met the query constraints. Or, after deleting a row with query constraints there are none left that match.

DB_NO_SERVER.....The server is not running.

DB_OK.....Normal return code - no error.

DB_ONLY_RECORD.....Returned when a record retrieved from the database with **FETCH_RECORD** is the only record in the table.

DB_OUT_OF_MEMORY.....An attempt to allocate memory on a file or table open failed. Close unneeded files and tables, then try again.

DB_PAGE_NOT_FOUND.....An internal database error has occurred. Close all files and tables and exit. Fatal error.

DB_QUERY_SYNTAX.....The query syntax requirements were not met on a query call. Check to be sure that there are no spaces within the query line and that the query operator is valid.

DB_RECNUM_ERR.....The record number given in a query, get or delete structure was not the number of a valid record in the database.

DeskMate Technical Reference

Database Resource

DB_RECORD_DELETED.....Temporary sort order error.

DB_RECORD_NOT_CURRENT.....On an **UPDATE_ROW** call with the `verify_flag` set to `VERIFY`, the old column values did not exactly match the data in the database. Fetch the new data, update the old column values in the structure, then try again.

DB_SAME_TABLE.....The same table handle was given in a **MERGE** call for both the source and destination tables.

DB_SERVER_BUSY.....The database server is busy. Try again after a delay. If the call to **beclient** had `TRUE` for the `box_flag` parameter this is returned when the user CANCEL's from a busy **RETRY/CANCEL** message box.

DB_SERVER_DIED.....The database server is up, but has been rebooted since the client's last request. The client's session is no longer valid. The client must close everything, invoke **not_client** and start over with a new **be_client** call.

DB_TABLE_FULL.....The maximum record number has been reached for the current table. The application must delete records and pack the table before adding more records.

DB_TABLE_LOCKED.....A data access/modification call was made for a table that is currently opened for a data definition. The table must be closed and re-opened for data access before the call will succeed.

DB_TABLE_NEEDS_UPDATE.....The table has been updated too much to return all changes in a table history structure. Redo any out-of-date-queries.

DB_TABLE_SHARED.....An attempt was made to access a table for data definition that is opened for data access/modification. You will have to wait until all user's have closed the table, then re-open it for data definition.

DB_TBL_NAME_ERR.....The table name in a table access structure does not match any of the table names in the directory for the given file handle.

DB_TIMEOUT.....The database server timed out.

DB_TOO_MANY_OPEN_FILES....A request to open a data base file was made when the maximum number were already open. You will have to wait until a file is closed before opening another one.

DB_UNIMPLEMENTEDAn undefined function code was given.

DB_UNIQ_NOT_SORT.....An attempt was made to define a new sort order which did not have the column currently specified as unique as it's first element.

DB_WRITE_PROTECTED.....An updated database cannot have changes saved because it is on a write protected disk.

Desk Executive

"Desk Executive"

Table of Contents

General Description/Notes	7- 1
Autoload Resource Initialization Calls	
autoload_bind_end ...Release the autoload resource.....	7- 4
autoload_bind_init ..Initializes autoload resource.....	7- 5
Database Resource Initialization Calls	
Database Defines	7- 6
db_bind_end ...Terminates bindings to database resource....	7- 7
db_bind_init ..Initializes all database resources.....	7- 8
db_bind_read ..Binds to 1989 Database Read Resource.....	7- 9
db_end_read ...Terminates bind to 1989 DB Read Resource....	7-11
END_DBUnbinds 1989 DB resources.....	7-12
INIT_DBBinds 1989 DB resources	7-13
Desk Executive Defines.....	7-14
dm_acc_runRuns an accessory.....	7-15
dm_bind_resourceDetermines if resource is in memory...	7-17
dm_clear_resourceRemoves resource from memory.....	7-18
dm_compatCompatibility for 1988 DM 03.02.0x....	7-19
dm_dir_isithereChecks existance of file.....	7-21
dm_exec_dont_shedPrevents code shedding.....	7-22
dm_file_searchSearches entire for file.....	7-23
dm_get_memAttempts to allocate DOS memory.....	7-24
dm_inquire_critical_err .Gets the last critical error.....	7-25
dm_inquire_desk_verReturns version number of DESK.EXE....	7-26
dm_inquire_dmconfigGets the contents of DMCONFIG.....	7-27
dm_inquire_floppyDetermines if a drive can be read.....	7-28
dm_inquire_productGets the product information.....	7-29
dm_inquire_programReturns current application names.....	7-30
dm_inquire_resReturns resource names.....	7-31
dm_inquire_res_verReturns version number of a resource..	7-32
dm_inquire_taskidReturns task information.....	7-33

dm_SetNextAppSets name and parameter string..... 7-34
dm_SetOtherTaskSets name of task for switch..... 7-35
dm_task_countTells number of tasks in mem..... 7-36
dm_temp_resourceLoads resource in temporary mode..... 7-37
dm_yieldAllow another task to run..... 7-38

Core Services Resource(CSR) Initialization Calls

csr_access_endTerminates current active access..... 7-39
csr_access_initInitializes CSR for new user access... 7-40
csr_endTerminates application bindings to CSR 7-41
csr_form_bind_endTerminates bind to near form resource 7-42
csr_form_bind_initBind to near form resource..... 7-43
csr_get_versionGets the version number of the CSR.... 7-44
csr_initInitialize application bindings to CSR 7-45
dmcsr_bind_endTerminates use of CSR routines..... 7-46
dmcsr_bind_initUsed to bind to the CSR..... 7-47
fform_bind_end.....Terminates far form resource..... 7-48
fform_bind_int.....Bind to far form resource..... 7-49
guf_bind_endTerminates application bindings to GUF 7-50
guf_bind_initInitialize application bindings to GUF 7-51
prguf_bind_endTerminates low-level GUF bindings..... 7-52
prguf_bind_initSets up low-level GUF bindings..... 7-53

Spell Resource Initialization Calls

spell_bind_end.....Terminates interface to spell checker. 7-54
spell_bind_init.....Initializes a spell checker session... 7-55

Thesaurus Resource Initialization Calls

thes_bind_end.....Terminates interface to thesaurus..... 7-56
thes_bind_init.....Initializes a thesaurus session..... 7-57

DeskMate Technical Reference

Desk Executive

General Description/Notes

Executive Specification

The executive is a small, resident module which provides the operating system services for DeskMate. It is responsible for the loading and releasing of resources, applications and accessories, managing the clipboard, and providing a communication system for task switching.

The executive dynamically links resources to applications, allowing multiple applications to share a resource.

Since the desk executive is responsible for loading and unloading all of the DeskMate resources, all of DeskMate's initialization calls are included in this section. See below for a more complete description.

Method of Operation

When loaded, the executive will setup the machine state to allow DeskMate to run efficiently. It will then load the core services resource. If the load was successful, the executive will proceed to load the default application and then pause.

While in its paused state the executive can be called via int E0h. It will handle the interrupt and then resume the paused state. After the current application exits, control is returned to the executive which then evaluates the next application to be run, and based on the information, either loads another application, or resets the machine to its pre-DeskMate state, and exits to DOS.

The executive is also able to start a new task, activating task switching between two applications. It will keep track of which task is which and handle coordination of DeskMate where multiple tasks are loaded.

Summary of changes to DESK.EXE for 1989

Applications may check for the 1989 version DESK.EXE through **dm_inquire_product**. If the **DM_VERSION** bit is set in the return value, then DESK.EXE is version 1989 or later. To get the specific version number call **dm_inquire_desk_ver**.

dm_load_resource and **dm_free_resource** are now available to applications.

The calls for the intelligent help manager are as specified in the intelligent help manager section of this manual.

dm_get_mem may be used to force desk to remove all unused resources, by asking for FFFF paragraphs of memory. The desk executive will remove all unused resources, allocate no memory, and return error.

If a call to **dm_load_resource** finds a resource that was loaded with **dm_temp_resource** that is in code shed space or in the other task's area, it will not bind to it. If the temp resource is unused, it will remove it from memory and try to load it again in a different area. If the temp resource is in use, a not enough memory message box will be displayed and an error returned to the calling application.

DeskMate Technical Reference

Desk Executive

Initialization

Initialization of the CSR (Core Resource Services) involves setting up an interface between the application and the CSR and initializing the CSR for use by the application. The bindings for the CSR are included in the DM.LIB and DMMED.LIB libraries. Within the binding for the **csr_init** routine is the process that sets up the interface between the application and the CSR. Therefore, the **csr_init** routine **MUST** be called prior to calling any other CSR routine.

The GUF (General User Functions) resource contains many functions of DeskMate. Initialization of the GUF resource involves setting up an interface between the application and GUF. The bindings for GUF are included in the DM.LIB and DMMED.LIB libraries. The **guf_bind_init** routine **MUST** be called prior to calling any other GUF routine. A list of the calls contained in the GUF resource are listed in the special notes section of the **guf_bind_init** call.

The other initialization routines which are required by the separate resources, AUTOLOAD, DATABASE, SPELL, and THESAURUS, are described here as well as in their respective sections.

Assembly language interface to the Desk Executive

For the following calls, the arguments are passed on the stack with bp pointing to the first argument, [bp+2] points to the second argument, etc. The ES register must be set to the segment which contains the arguments.

Library Call	Value for AX
_dm_SetNextApp	DM_EXEC_SETNEXTAPP
_dm_SetOtherTask	DM_EXEC_SETOTHERTASK
_dm_exec_dont_shed	DM_EXEC_DONT_SHED
_dm_yield	DM_EXEC_YIELD
_dm_task_count	DM_EXEC_TASK_COUNT
_dm_get_clipboard_info	DM_EXEC_CLIP_GETINFO
_dm_set_clipboard_info	DM_EXEC_CLIP_SETINFO
_dm_clear_clipboard	DM_EXEC_CLIP_CLEAR
_dm_inquire_product	DM_EXEC_INQ_PRODUCT
_dm_inquire_taskid	DM_EXEC_TASKID
_dm_inquire_critical_err	DM_EXEC_LASTCRIT
_dm_inquire_program	DM_EXEC_TASKNAME
_dm_inquire_res	DM_EXEC_RESOURCES
_dm_inquire_floppy	DM_EXEC_FLOPPY
_dm_inquire_desk_ver	DM_EXEC_VERSION
_dm_get_mem	DM_EXEC_GET_MEM

For the following calls, the argument is passed in DI. The segment is ES.

Library Call	Value for AX
_dm_inquire_dmconfig	DM_EXEC_INQ_DMCONFIG
_dm_dir_isithere	DM_DIR_ISITHERE

For the following calls, the first argument is passed in DX and the second argument, if any, is passed in BX. The segment for both arguments are ES.

DeskMate Technical Reference

Desk Executive

Library Call

_dm_load_resource
_dm_temp_resource
_dm_free_resource
_dm_bind_resource
_dm_clear_resource
_dm_inquire_res_ver

Value for AX

DM_EXEC_LOAD_RES
DM_EXEC_TEMP_RES
DM_EXEC_FREE_RES
DM_EXEC_BIND_RES
DM_EXEC_CLEAR_RES
DM_EXEC_INQ_RESVER

DeskMate Technical Reference
Desk Executive
Autoload Resource Initialization Calls

autoload_bind_end ()

autoload_bind_end enables applications to release the autoload resource.

Input Parameters

None.

Return Value

None.

Special Notes

If an application calls **autoload_bind_init**, it must call **autoload_bind_end** when finished with the autoload resource.

DeskMate Technical Reference

Desk Executive

autoload_bind_init ()

autoload_bind_init sets up the bindings for the autoload resource, enabling the application to use functions in the autoloader.

Input Parameters

None.

Return Value

<int>

DM_ERROR if the call was unsuccessful.

Special Notes

autoload_bind_init should be made before calling any autoloader calls.

DeskMate Technical Reference

Desk Executive

Database Defines

Only the binding defines are listed here, the remainder of the Database structures and defines are listed in the Database Resource section.

```
/* defines for initialization binding and unbinding */
BIND_READ_ONLY 0          /* for 89 app support */
BIND_ALL 1          /* for server/88 network app support */
BIND_UPDATE 2          /* bind update resource */
BIND_BUILD 3          /* bind build and update resources */
BIND_NETWK 4          /* bind network resource */
READ_ONLY 5          /* parameter to db bind read */
READ_WRITE 6          /* parameter to db_bind_read */
```

**DeskMate Technical Reference
Desk Executive**

Database Resource Initialization Calls

db_bind_end ()

db_bind_end terminates the bindings to the the database manager resource.

Input Parameters

None.

Return Value

None.

Special Notes

db_bind_end is to be used with **db_bind_init** for 1988 DeskMate 03.02.0x compatability. **db_bind_end** should not be used with **db_bind_read**. **db_bind_end** should be called when no more database support is required by the application, and before **csr_end**.

DeskMate Technical Reference

Desk Executive

db_bind_init ()

db_bind_init causes the Read, (DBREAD.RES), Update, (DBUPDATE.RES), and Build, (DBBUILD.RES) resources to be loaded into memory.

Input Parameters

None.

Return Value

DB_OK if loading was successful.

DM_ERROR if loading was unsuccessful.

Special Notes

db_bind_init is to be used with **db_bind_end** for 1988 DeskMate 03.02.0x compatability. **db_bind_init** should be called prior to making any database manager requests, and after the **csr_init** call.

Example

```
if ( db_bind_init < DB_OK )  
    /* do not make any database calls */
```

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

db_bind_read (ReadFlag)
int ReadFlag;

db_bind_read allows the application to utilize the 1989 Database Read Resource. **db_bind_read** binds to the 1989 Database Manager Resource DMDB.R89 and the database read resource DBREAD.RES.

Input Parameters

ReadFlag is **READ_ONLY** if the database is to be used for read-only.

ReadFlag is **READ_WRITE** if the database is to be used for updating/building. When specifying **READ_WRITE** the application must call **db_mgr(INIT_DB, BIND_UPDATE)** or **db_mgr(INIT_DB, BIND_BUILD)** depending upon the desired functionality.

Return Value

<int>

DB_OK if successful.

DB_READ_ONLY if only the read-only function is available and the application asked for **READ_WRITE**.

DM_ERROR if the desk executive was unable to find the resource(s) or there was not enough memory to load the resource(s).

Special Notes

db_bind_read should only be used with **db_end_read**.

If the request is made for more than read-only, and the return code is **DB_READ_ONLY**, the application may still continue, however only the read-only functions may be called. The application must still unbind to the resource via **db_end_read**. **DM_ERROR** will cause an error message to appear on the screen. **DB_READ_ONLY** will not cause an error message.

The calls listed below are contained in the Database Read Resource:

CLOSE_FILE	CLOSE_TABLE	COUNT_RECORDS
END_DB	FETCH_RECORD	FIRST_RECORD
GET_COL_INFO	GET_COLUMN_NAMES	GET_INDEX_INFO
GET_MAX	GET_MIN	GET_PAGE_SETUP
GET_RECORD_LOCATION	GET_TABLE_NAMES	INIT_DB
LAST_RECORD	MORE_RECORDS	NEXT_RECORD
OPEN_FILE	OPEN_TABLE	PREV_RECORD
SETUP_QUERY		

DeskMate Technical Reference

Desk Executive

Example

```
/* bind to the 1989 DeskMate 03.03.0x read resource */
if (db_bind_read( READ_WRITE ) == DB_OK)
{
    /* try to bind to the update resource */
    if (db_mgr( INIT_DB, BIND_UPDATE ) == DB_OK)
    {
        .      /* calls to update the database */
        .
        /* unbind to the database update resource */
        db_mgr( END_DB, BIND_UPDATE );
    }
}
/* unbind to the database read resource */
db_end_read();
}
```

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

db_end_read ()

db_end_read terminates the binding to the Database Read Resource. **db_end_read** unbinds to the 1989 Database Manager Resource DMDB.R89 and the database read resource DBREAD.RES.

Input Parameters

None.

Return Value

None.

Special Notes

db_end_read should only be used with **db_bind_read**. **db_end_read** should be called when no more database read support is required by the application.

Example

```
/* bind to the 1989 DeskMate 03.03.0x read resource */
if (db_bind_read( READ_WRITE ) == DB_OK)
{
    /* try to bind to the update resource */
    if (db_mgr( INIT_DB, BIND_UPDATE ) == DB_OK)
    {
        .
        .      /* calls to update the database */
        .
        .
        /* unbind to the database update resource */
        db_mgr( END_DB, BIND_UPDATE );
    }
    /* unbind to the database read resource */
    db_end_read();
}
```

See Also

db_bind_read()
INIT_DB

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

END_DB (Resource)

int Resource;

END_DB will unbind database resources from the Database Resource Manager.

Input Parameters

Resource if **BIND_UPDATE** the application may no longer access the database update functions.

Resource if **BIND_BUILD** the application may no longer access the database build functions.

Resource if **BIND_ALL** the application may no longer access any of the database functions.

Return Value

None.

Special Notes

END_DB must be called before **db_end_read**.

Example

```
/* unbind to the build resource */  
erc = db_mgr ( END_DB, BIND_BUILD )
```

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

INIT_DB (Resource)

int Resource;

INIT_DB will bind database resources to the Database Resource Manager.

Input Parameters

Resource if BIND_UPDATE the application may access the database update functions.

Resource if BIND_BUILD the application may access the database build functions.

Resource if BIND_ALL the application may access all of the database functions.

Return Value

DB_OK if successful.

DB_OUT_OF_MEMORY if the resource requested could not be loaded.

DB_READ_ONLY if an attempt was made to load a resource and db_bind_read bindings returned a DB_READ_ONLY.

Special Notes

db_bind_read must return DB_OK before calling INIT_DB. A return of DB_OUT_OF_MEMORY will cause an error message to appear to the user. DB_READ_ONLY will not display an error message.

The calls listed below are contained in the Database Build and Update Resources:

ALTER_TABLE CREATE_TABLE DROP_TABLE	Build Resource	
	COPY_LAYOUTS	CREATE_FILE
	DEFINE_INDEX	DROP_INDEX
	PACK_TABLE	
ADD_ROW DELETE_ROW SAVE_PAGE_SETUP	Update Resource	
	ADD_ROW_NO_FLUSH	DELETE_MULTI_ROWS
	END_TEMP_SORT	MERGE
	TEMP_SORT	UPDATE_ROW

Example

```
/* call db bind read */  
/* see db bind read */  
/* bind to the build resource */  
erc = db_mgr ( INIT_DB, BIND_BUILD )
```

See Also

END_DB

DeskMate Technical Reference

Desk Executive

Desk Executive Defines

```

/*-----*/
/*
/* DMEEXEC.H contains all of the Desk Executive defines */
/*-----*/

/*-----*/
/*
/* The following are the defines that define function parameter */
/* constants and return values used in calls to the exec. */
/*-----*/

DM_OK          1      /* returned when exec request was successful */
DM_ERROR       -1     /* returned when an error in exec occurred */
DM_NOT_ALLOWED -2     /* returned when an illegal task sw call */
DM_DEFAULT     2      /* from DM_EXEC_INQ_DMCONFIG means DESK */
                /* assigned */
DM_EXISTS      -2     /* value returned when trying to load a */
                /* resource that already existed on an */
                /* 1988 DeskMate System. */

/* Bits set by DM_EXEC_INQ_PRODUCT */
DM_RUN_FLAG    4      /* This is the runtime version */
DM_COMPAT_FLAG 8      /* 1989 Runtime running with */
                /* 1988 DeskMate */
DM_SHELL_FLAG  0x010  /* Desktop only "shell" product */
DM_DESKLINK    0x100  /* DeskLink active */
DM_TASKSWITCHING 0x200 /* Task switching is active */
DM_VERSION     0x8000 /* DM_EXEC_INQ_VERSION call */
                /* is available */

/* These are the different clipboard type */
/* available to the applications. */
CLIP_EMPTY     0
CLIP_TEXT      1
CLIP_BITMAP    2
CLIP_WORKSHEET 3
CLIP_FILER     4
CLIP_PAINT     5
CLIP_DRAW      6
CLIP_DRAW_TEXT 7
CLIP_MUSIC     8
CLIP_CALENDAR  9

ACC_BY_NAME    0xff /* to give the name of the accessory */

```

DeskMate Technical Reference

Desk Executive

dm_acc_run (Accessory)
int Accessory;

dm_acc_run runs an accessory.

Input Parameters

Accessory is the code for the accessory to run.

Return Value

None.

Special Notes

Range checking should not be done by an application since the accessory list might change and grow on different version of the software. For example do *not* do tests such as:

```
if (Accessory <= ACC_SETUP )
    dm_acc_run (Accessory );
```

The following are the different accessories available along with their associated codes. The codes can be found in CSRBASE.H:

```
ACC_CALC          0x00
/* 0x01 must remain undefined due to MS-DOS parameter limitations */
ACC_NOTEPD        0x02
ACC_CORKBRD       0x02
ACC_CAL           0x03
ACC_CLIPBD        0x04
ACC_COLORS        0x05
ACC_COMM          0x06
ACC_DATTIM        0x07
ACC_MOUSE         0x08
ACC_PRINTR        0x09
ACC_HELP          0x0A
ACC_PGSETUP       0x0B
ACC_PHONE         0x0C
ACC_DESKLINK      0x0D
ACC_VIDEO         0x0E
ACC_TASKSWITCH    0x0F
ACC_SPELL         0x10
ACC_ALARM         0x11
ACC_SETUP         0x12
ACC_TALK          0x13
ACC_TODO          0x14
ACC_MORE          0x15

/* resource referenced by name, not index */
ACC_BY_NAME       0xFF
```

dm_acc_run is run following an event_appl, appl_access event. **dm_acc_run** is usually run using the event .x variable in an EVENT structure as the **Accessory** parameter.

DeskMate Technical Reference

Desk Executive

If ACC_BY_NAME is specified then **dm_acc_run** requires two parameters. The second parameter being the name of the accessory the application wishes to run. For example to run the phone accessory the call would be made as follows:

```
dm_acc_run ( ACC_BY_NAME, "PHONE" );
```

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dm_bind_resource (pResName, pResEntry)

char *pResName;

char *pResEntry;

dm_bind_resource is used to determine if a resource is in memory.

Input Parameters

pResName is a near pointer to the resource name.

pResEntry is a near pointer to a far pointer to a procedure.

Return Value

NULL if resource is not already loaded.

DM_OK if the resource is loaded and **pResEntry** receives a pointer to the resource's binding entry point.

DM_ERROR if the DeskMate Core Services Resource CSR is busy initializing or terminating, try again.

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dm_clear_resource (pResName)
char *pResName;

dm_clear_resource removes a resource from memory if its use count is zero.

Input Parameters

pResName is a near pointer to the resource name.

Return Value

NULL if resource is not already loaded, it is in use, or was autoloaded.

DM_OK if the resource was removed from memory.

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dm_compat (pName)
char *pName;

When **dm_compat** is called when running under the 1988 version of the desk executive, the clipboard is set to the maximum size, 8K or 16K depending upon the memory size of the machine, **dm_SetNextApp** is called to run the name specified by **pName**, and **DM_COMPAT_FLAG** is returned to the application as described below.

Input Parameters

pName is the name of the file to execute, including the .EXE.

Return Value

The following chart summarizes the possible situations under which the application may run and what **dm_compat** will do in each situation after calling **dm_inquire_product** and calling **dm_task_count** if under the 1988 desk executive:

If the desk executive version is 1989, from DeskMate:

DM_VERSION on
DM_RUN_FLAG off
DM_COMPAT_FLAG off

If the desk executive version is 1989, runtime run from DOS:

DM_VERSION on
DM_RUN_FLAG on
DM_COMPAT_FLAG off

If the desk executive version is 1988 and desk is not Task Switched:

DM_VERSION off
DM_RUN_FLAG off
DM_COMPAT_FLAG on

If the desk executive version is 1988 and desk is Task Switched:

DM_VERSION off
DM_RUN_FLAG off
DM_COMPAT_FLAG off

If the desk executive version is 1989 runtime running on 1988 desk:

DM_VERSION on
DM_RUN_FLAG on
DM_COMPAT_FLAG on

Example

```
product_info = dm_compat("VENDOR.EXE");  
if ((product_info & DM_VERSION) == 0)  
{  
    /* we're running on 1988 desk */  
    if ((product_info & DM_COMPAT_FLAG) == 0)  
    {  
        /* bind to csr */  
        csr_init();  
        /* display error message can't run task-switched */  
        display "Cannot run while task-switched."  
    }  
}
```

DeskMate Technical Reference

Desk Executive

```
        /* un-bind to csr */  
        csr_end();  
    }  
    exit();  
} /* running on a 1989 DeskMate or Runtime */
```

See Also

DeskMate Development Guide Compatibility Issues

DeskMate Technical Reference

Desk Executive

dm_dir_isithere (pName)
char *pName;

dm_dir_isithere determines whether a specific file in a specific path exists.

Input Parameters

pName is a pointer to a null terminated pathname.

Return Value

Zero if not found.

One (1) if file was found.

Special Notes

dm_dir_isithere will test any removable drives first to ensure there is a disk in the drive. This is especially helpful when an application does not want a critical error to occur when looking for a file. There is a case where a critical error will occur, when only a filename is specified along with a drive, and there is no disk in the drive.

dm_dir_isithere is used primarily with a file appended to the DMCONFIG variable.

DeskMate Technical Reference

Desk Executive

dm_exec_dont_shed ()

dm_exec_dont_shed tells the executive not to code shed the current application.

Input Parameters

None.

Return Value

None.

Special Notes

If an application cannot have a portion of itself moved and replaced from the .PDM file on disk, when an accessory is invoked, then the application should call **dm_exec_dont_shed**, in its startup code.

In 1988 DeskMate 03.02.0x, there were a set number of fixups which could occur to the application when returning from running an accessory. In 1989 DeskMate 03.03.0x there is no limit imposed by the desk executive. Therefore medium model applications should check for the version of DeskMate, and call **dm_exec_dont_shed** in instances where too many fixups would occur.

Example

```
dm_exec_dont_shed();
```

See Also

dm_inquire_product()

The DESKHDR documentation in the Tools and Utilities manual.

DeskMate Technical Reference

Desk Executive

dm_file_search (pFilename, pPathbuffer, bFloppySearch)

char *pFilename;

char *pPathbuffer;

int bFloppySearch; ←

This function is prototyped twice in Dmdecl.h. The first prototype says "int bOnlyFlop". Second says this. Second is wrong.

PROMPT #1

dm_file_search will first inform the user that FILENAME could not be found and that it will search the entire system for it. If the user presses OK, it will then search all directories on all available disk drives starting first with the floppy drives, then searching hard drives and redirected drives.

If the user cancels the second prompt will appear.

PROMPT #2

The second prompt asks the user to place a disk containing the file requested in any floppy disk drive. If the user presses OK, all floppy drives are searched. If the file is not found, Prompt #2 is re-displayed.

If the user cancels, **dm_file_search** returns a zero(0) to the application.

If the file is found in either search, the full path is placed in **Pathbuffer** and a 1 is returned from **dm_file_search**.

Input Parameters

pFilename is a pointer to the null-terminated filename to be found (the filename must be upper-case).

pPathbuffer is a pointer to a buffer into which the full path including the filename will be placed if the file is found.

bFloppySearch can be set to one of four values as explained in the table below.

Value	Functionality
0	Complete system search with prompts
1	Floppy only search (prompts before search #2 above)
2	Complete system search with NO prompts
3	Floppy only search with NO prompts

Return Value

<int>

Zero (0) if the file was not found.

One (1) if the file was found, **Pathbuffer** contains full path (the disk containing the file is still in the drive).

DM_ERROR if media failure or critical error that user cancelled.

2 and 3 do not work as advertised, at least under DM302 [3=2=1]. Only 0 or 1 should be used.

DeskMate Technical Reference

Desk Executive

Special Notes

pPathbuffer may contain invalid information if the file was not found.

Prompting will work correctly when DeskMate is active and when DeskMate is inactive (ie., character prompts are displayed).

If no extension exists on file, priority of files found in a single directory are PDM, COM, EXE, BAT. Search will stop as soon as the first match has been made.

Maximum of ten directory levels will be searched (requires about 1000 bytes of stack space).

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dm_get_mem (nPara)

int nPara;

dm_get_mem attempts to allocate DOS memory. **dm_get_mem** works like an INT 21h function 48h, except that the desk executive will attempt to remove unused resources from memory if there is not already enough memory for the allocation.

Input Parameters

nPara is an integer specifying a number of paragraphs.

Return Value

NULL if not enough memory.

The segment address of memory allocated if successful.

See Also

DOS INT 21h function 48h

DeskMate Technical Reference
Desk Executive

dm_inquire_critical_err ()

dm_inquire_critical_err returns the last critical error encountered.

Input Parameters

None.

Return Value

The number of last critical error.

Special Notes

The error numbers are DOS int 24 error numbers.

DeskMate Technical Reference

Desk Executive

dm_inquire_desk_ver ()

dm_inquire_desk_ver returns the version number of DESK.EXE.

Input Parameters

None.

Return Value

The version number of the desk executive. The High Order byte is the Major version number. The Low Order byte is the Minor version number. The table below shows what numbers are returned for different versions of DeskMate.

DeskMate Version	Desk Executive Version	Release
03.03.00	55.00	SL
03.03.00	55.00	TL
03.03.01	65.00	FT1
03.03.01	66.00	2500 XL
03.03.01	66.00	Retail
03.04.00	67.03	Runtime '89
03.04.01	68.12	1000 RL
03.05.00	68.17	1000 RL cutin
03.05.00	68.18	TL/3
03.05.00	68.19	Retail
03.05.00	68.21	Runtime '90

Special Notes

dm_inquire_desk_ver is only available when running DeskMate 03.03 and later.

DeskMate Technical Reference

Desk Executive

dm_inquire_dmconfig (pString)
char *pString;

dm_inquire_dmconfig gets the contents of the environment variable DMCONFIG.

Input Parameters

pString is a pointer to a character string where the environment configuration is to be stored.

Return Value

DM_OKAY if user set the DMCONFIG environment variable.
DM_DEFAULT if desk set it.

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dm_inquire_floppy (Drive)

int Drive;

dm_inquire_floppy checks to see if the specified floppy drive can be read.

Input Parameters

Drive is the floppy drive to check. 0 = A, 1 = B, etc.

Return Value

NULL if the drive is floppy drive with no disk in drive.

One (1) if the drive is a floppy drive with a disk in the drive.

Greater than 1 if drive is not a floppy.

DeskMate Technical Reference

Desk Executive

dm_inquire_product ()

dm_inquire_product returns the product information for DeskMate including:

1. DeskMate or Runtime.
2. WRKGROUP availability.
3. Taskswitching capability.

Input Parameters

None.

Return Value

<int>

DM_RUN_FLAG if this is vendor runtime.

DM_TASKSWITCHING if task switching is allowed.

DM_DESKLINK if wrkgroup installed.

DM_VERSION if DeskMate 03.03 or later; **dm_inquire_desk_ver** is available, to get the actual version.

DM_COMPAT_FLAG is returned when the 03.03 runtime is running under 03.02 DeskMate. The 03.03 runtime sets this bit if it finds it is running under the 03.02 desk executive. The 03.02 desk executive never returns this bit set.

Two (2) if not runtime.

Combinations are returned ORed together.

Example

```
int    Product;

Product = dm_inquire_product();
if ( Product & DM_VERSION )
    /* DeskMate Version 3.03 or later */
```

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dm_inquire_program (pName)
char *pName;

dm_inquire_program returns the name of the current application or accessory.

Input Parameters

pName is a near pointer to a buffer to receive the program name. The buffer should be at least 13 characters long.

Return Value

The buffer is filled with the application or accessory name.

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dm_inquire_res (pBuffer)
char *pBuffer;

dm_inquire_res fills the buffer with the names of resources currently in memory and indicates if they are loaded or preloaded.

Input Parameters

pBuffer is a near pointer to a buffer to receive the resource information.

Return Value

The buffer is filled with the resource information, and the number of resources is returned.

Special Notes

Each entry takes ten bytes. Nine bytes for the filename and one byte for the autoload priority. There is a maximum of 32 resources allowed in the system at any one time.

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dm_inquire_res_ver (pResName)
char *pResName;

dm_inquire_res_ver returns the version number for a resource.

Input Parameters

pResName is a near pointer to a resource name.

Return Value

The version number of the resource.

Zero (0) if the resource is not registered or did not register with a version number.

DeskMate Technical Reference

Desk Executive

dm_inquire_taskid ()

dm_inquire_taskid returns an indicator for which program is in control. It is used by the shared database routines.

Input Parameters

None.

Return Value

<int>

One (1) for first task.

Two (2) for alternate task.

Three (3) for an accessory.

Example

```
int taskid;  
taskid = dm_inquire_taskid();
```

DeskMate Technical Reference

Desk Executive

dm_SetNextApp (lpName, lpParam)

char far *lpName;

char far *lpParam;

dm_SetNextApp sets the name of the executable application and it's parameter string that is to be executed when the current application is terminated.

Input Parameters

lpName is a far pointer to the path name of the application to be executed. If **lpName** is a pointer to a null string DeskMate exits.

lpParam is a far pointer to the parameter list.

Return Value

DM_OK if successful.

DM_ERROR if failure.

Special Notes

dm_SetNextApp can fail if the user is currently task switched to a non-DeskMate application and the currently running application is attempting to set the next application to another non-DeskMate application.

To set up this call to run .COM files:

- 1) It is the calling application's responsibility to place 0Dh(ascii carriage return) at the end of a non null parameter string.
- 2) It is safest to make the first character of the parameter list a space because some older .COM applications ignore the first character of the parameter list.

For DeskMate to exit **lpName** should be a pointer to a null string *not* a null pointer.

The parameters **lpName** and **lpParam** may point to automatic local, non-static, on the stack buffers because **dm_SetNextApp** copies the contents of the parameters using the pointers, to internal buffers in the Desk Executive (DESK.EXE). This is done because the data must be used AFTER your application exits.

DeskMate Technical Reference

Desk Executive

dm_SetOtherTask (lpName, lpParam)

char far *lpName;
char far *lpParam;

dm_SetOtherTask sets the name of an executable task and its parameter string. The task is to be executed when a task switch occurs.

Input Parameters

lpName is a far pointer to the path name of the task to be executed. If **lpName** is a pointer to a null string DeskMate exits.

lpParam is a far pointer to the parameter list.

Return Value

None.

Special Notes

dm_SetOtherTask should only be called if **dm_task_count** returns one (1).

To set up this call to run .COM files:

- 1) It is the calling application's responsibility to place 0Dh(ascii carriage return) at the end of a non null parameter string.
- 2) It is safest to make the first character of the parameter list a space because some older .COM applications ignore the first character of the parameter list.

For DeskMate to exit **lpName** should be a pointer to a null string *not* a null pointer.

dm_SetOtherTask does not update the task count. The task is executed through **dm_yield**.

DeskMate Technical Reference

Desk Executive

dm_task_count ()

dm_task_count provides information about how many tasks are in memory.

Input Parameters

None.

Return Value

The number of active tasks in memory.

DeskMate Technical Reference

This page intentionally left blank.

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dm_temp_resource (pResName, pResEntry)

char *pResName;

char *pResEntry;

dm_temp_resource attempts to load a resource in "temporary mode" and return an entry to its binding point.

Input Parameters

pResName is a near pointer to the resource name.

pResEntry is a near pointer to a far pointer to a procedure.

Return Value

NULL if the resource is not loaded.

DM_OK if the resource loaded, **pResEntry** received the address of the resource's binding entry point.

Special Notes

The resource may be loaded in code shed space or in part of the other task's memory area. Resources must be freed before the accessory exits or before an application task switches.

DeskMate Technical Reference

Desk Executive

dm_yield ()

dm_yield allows one application to yield to another in response to a task switch event.

Input Parameters

None.

Return Value

DM_OK if a alternate task yielded as expected.

DM_ERROR if a non recoverable error occurred during the alternate task.

Special Notes

In general, task switching allows the DeskMate version 3, and Professional DeskMate users (not supported in Personal DeskMate), to have two applications active at the same time, switching between them on user command. The only restrictions on what two applications can be in memory at the same time is that one must be a DeskMate version 3, or Professional DeskMate application (non-DeskMate apps can use the task switcher) and that only one copy of the default application may be loaded. Task switching is not supported in the DeskMate Runtime environment.

The application will make the **dm_yield** call in response to an EVENT_APPL with parameter APPL_TASK_SWITCH. The rule of thumb should be any time an accessory can be run, task switching can occur. This means NO task switching in accessories, dialog boxes or message boxes. When the application gets this event, it performs processing that places the application into a stable state, such as removing any event interpreters, event drivers and interrupt vectors then calls **dm_yield**. If the application must keep interrupt vectors active during the running of an accessory, then the interrupt vector must be located above the applications split address, as described in the DeskHdr section of the Tools and Utilities manual.

On a normal return (DM_OK), from **dm_yield**, the application should redraw its entire screen including the menu bar. It should also check the clipboard for a change in contents and register it's page setup information with the core (since the other application may have changed it).

On a abnormal return (DM_ERROR), the application must clean up and do an immediate exit (program termination).

Applications which hook interrupt vectors must be aware that vectors are not swapped between DeskMate applications. This causes problems with 'C' applications that use floating point emulation because these vectors are changed when a second application hooks them.

DeskMate Technical Reference Desk Executive

Core Services Resource (CSR) Initialization Calls

1989 DeskMate 03.03.0x ONLY

csr_access_end ()

csr_access_end is used to terminate the use of the current active access session, which in turn returns to the previous access session.

csr_access_end must be made only if **csr_access_init** was made.

Input Parameters

None.

Return Value

None.

See Also

csr_access_init()
csr_init()
csr_end()
dmcsr_bind_end()
dmcsr_bind_init()

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

int **csr_access_init** ()

csr_access_init initializes the CSR for a new user access session. This gives the effect of creating a new base window with default settings without destroying the previous access session state.

Each application can have more than one access session. The amount is dependent on the number of window and hotspot created under each of the access session.

Input Parameters

None.

Return Value

The handle of the base window. (new current active window.)

See Also

csr_access_end()
csr_end()
csr_init()
dmcsr_bind_end()
dmcsr_bind_init()

DeskMate Technical Reference

Desk Executive

csr_end ()

csr_end terminates the application interface to the CSR.

Input Parameters

None.

Return Value

None.

Special Notes

csr_end must be called at the end of each application.

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

csr_form_bind_end ()

csr_form_bind_end terminates the form manager loaded by **csr_form_bind_init**.

Input Parameters

None.

Return Value

<int>

One (1) if the resource was found and released.

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

csr_form_bind_init ()

csr_form_bind_init loads, executes, and links to the form manager resource as a near form manager usage. This is the same functionality provided in the 1988 DeskMate 03.02.0x.

Input Parameters

None.

Return Value

<int>

DM_ERROR if the form manager could not be loaded.

DM_EXISTS if the application is running on DeskMate 03.02.0x.

← Wrong. Causes system lockup under DeskMate prior to 3.03. Must check version before making this call.

DeskMate Technical Reference

Desk Executive

csr_get_version ()

csr_get_version returns the version number of the currently loaded DMCSR.RES.

Input Parameters

None.

Return Value

<int>

The version number of the currently loaded DMCSR.RES.

DeskMate Technical Reference

Desk Executive

csr_init ()

csr_init initializes the application interface to the CSR.

Input Parameters

None.

Return Value

<int>

The handle to the application's base window.

Special Notes

csr_init must be called at the beginning of the application before any other core service calls are made.

All CSR data and states are initialized as follows:

- The mouse pointer is defined as the default pointer image.

- No pointer regions are defined for the mouse pointer.

- The mouse pointer is turned off.

- All event and input device buffers are cleared.

- No main menu bar is defined.

- Only the default CSR components are available.

- Only the base window is defined.

- All help levels are defined as **HELP_NULL**.

- The configuration table modified flag is cleared.

- The world origin is set to 0, 0 and its extent is set to 8000, 5500.

- The viewport origin and extent are set to the full video display surface as defined by

 - vid_inquire_device**.

- The clip region origin is set to 0, 0 and its extent is set to 8000,5500.

- The cursor position is set to 0,0.

- The cursor is turned off.

- The cursor type is set to **VID_BLOCK_CURSOR**.

- The character attribute is set to **NORMAL**.

- The background color is set to **COLOR1** and the foreground color is set to **COLOR2**.

- The line type is set to **LINE_SOLID**, the line width to **LINE_THIN**, and the line color to

 - COLOR4**.

- The fill pattern is set to **PATTERN2**.

- The palettes are set according to the loaded video driver.

- All configuration states contained within **DMCSR.CFG** are setup.

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dmcsr_bind_end ()

dmcsr_bind_end is used to terminate the use of the CSR routines. **dmcsr_bind_end** must be made if **dmcsr_bind_init** was made, if not a use count on the CSR would be indicated when exiting DeskMate.

Input Parameters

None.

Return Value

None.

See Also

dmcsr_bind_init()
csr_access_end()
csr_access_init()
csr_end()
csr_init()

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

dmcsr_bind_init ()

dmcsr_bind_init is used to bind to the CSR. Its objective is to allow the user to call CSR routines without having to create an access session. (To be able to use the core without changing or creating another window.)

An example of if its use would be in the situation of swapping to a different video driver (mode).

Input Parameters

None.

Return Value

None.

See Also

dmcsr_bind_end()
csr_access_end()
csr_access_init()
csr_end()
csr_init()

DeskMate Technical Reference

Desk Executive

eform_bind_end ()

eform_bind_end terminates the forms resource loaded by **eform_bind_init** and unbinds any resources loaded by the enhanced (far) forms resource.

Input Parameters

None.

Return Value

None.

Special Notes

eform_bind_end requires the application to link with DeskMate 03.05 or later libraries (DM.LIB or DMMED.LIB).

DeskMate Technical Reference

Desk Executive

eform_bind_init (Font)

eform_bind_init loads and links to the forms resource as an enhanced (far) forms resource. This functionality is only available in for use in 1989 DeskMate 03.03.0x and later.

Input Parameters

<char>

If **Font** is **FALSE** the enhanced form manager will not support fonts. The font resource and enhanced video driver will not be loaded into memory.

If **Font** is **TRUE** the enhanced form manager will support fonts, and the enhanced video driver will be loaded into memory.

Return Value

<int>

DM_OK if everything was loaded correctly.

DM_ERROR if the forms resource could not be loaded..

Special Notes

eform_bind_init requires the application to link with DeskMate 03.05 or later libraries (DM.LIB or DMMED.LIB).

DeskMate Technical Reference
Desk Executive
GUF Resource Initialization Calls

guf_bind_end ()

In DeskMate 03.03.0x **guf_bind_end** terminates the applications bindings to both the low-level, (PRGUF.RES) and high-level, (DMGUF.RES) GUF resources.

In DeskMate 03.02.0x **guf_bind_end** terminates the applications bindings to the GUF resource.

Input Parameters

None.

Return Value

None.

Special Notes

guf_bind_end should be made when termination to the GUF resource(s) is desired by the application. Usually just prior to **csr_end**.

If the application makes this call it should *not* make the call to **prguf_bind_end**.

DeskMate Technical Reference

Desk Executive

guf_bind_init ()

In 1989 DeskMate 03.03.0x **guf_bind_init** sets up the applications bindings to both the low-level, (PRGUF.RES) and high-level, (DMGUF.RES) GUF resources.

In 1988 DeskMate 03.02.0x **guf_bind_init** sets up the bindings from the application to the GUF resource.

Input Parameters

None.

Return Value

DM_ERROR if the high-level GUF resource, DMGUF.RES could not be loaded.

Special Notes

guf_bind_init should be called when the user wishes to use any of the calls in either of the GUF resources, usually right after **csr_init**.

If the application makes this call it should *not* make the call to **prguf_bind_init**.

Functions included in the high level GUF, DMGUF.RES, which require **guf_bind_init**:

File I/O manager:

dlgbox_Generic	dlg_Run
dlgbox_SaveAs	dlg_Run_this(new for '89)
fil_close_error_msg	fil_create_error_msg
fil_lseek_error_msg	fil_menu_merge
fil_menu_new	fil_menu_open
fil_menu_page	fil_menu_quit
fil_menu_run	fil_menu_save
fil_menu_saveas	fil_open_error_msg
fil_read_error_msg	fil_write_error_msg
fil_already_exists	FillWithDir
msgbox_SaveModifiedData	msgbox_SaveChanges
Set_CPU_Speed	valid_filename
valid_filename_wild(new for '89)	

International Support Routines

get_month_strings

Printer Support

ptd_open

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

prguf_bind_end ()

prguf_bind_end terminates the applications bindings to the low-level GUF resource PRGUF.RES.

Input Parameters

None.

Return Value

None.

Special Notes

prguf_bind_end should be made when termination to the low-level GUF resource, PRGUF.RES is desired by the application. Usually just prior to **csr_end**.
prguf_bind_end is automatically made by **guf_bind_end**.

DeskMate Technical Reference

Desk Executive

1989 DeskMate 03.03.0x ONLY

prguf_bind_init ()

prguf_bind_init sets up the bindings from the application to the low-level GUF resource, PRGUF.RES.

Input Parameters

None.

Return Value

CSR_ERROR if the low-level GUF resource, PRGUF.RES could not be loaded.

Special Notes

prguf_bind_init should be called when the user wishes to use any of the calls in the low-level GUF resource, PRGUF.RES, usually right after **csr_init**.

prguf_bind_init is automatically made by **guf_bind_init**.

Calls contained in the PRGUF.RES resource:

Functions included in the low level GUF, PRGUF.RES, which require **prguf_bind_init**:

File I/O Manager

Current Disk	Delete File
dosENDFIND(new for '89)	dosFINDFIRST(new for '89)
dosFINDNEXT(new for '89)	dosSTARTFIND(new for '89)
fil_access	fil_close
fil_create	fil_dup
fil_force_close	fil_lock_bytes
fil_lseek	fil_open
fil_read	fil_read_far(new for '89)
fil_unlock_bytes	fil_write
fil_write_far(new for '89)	get_free_space
Get_Directory	get_drive_map
is_Floppy	Path_Expand
qfn(new for '89)	Rename_File
Select_Disk	valid_drive

Environment Manager

env_add	env_close
env_delete	env_get
env_open	env_replace
env_write	env_init_disk_labels
env_get_disk_labels	

International Support Routines

compare_chars	compare_chars_ins
get_intl	str_upper
to_upper	

DeskMate Technical Reference
Desk Executive
Spell Resource Initialization Calls

void spell_bind_end ()

spell_bind_end terminates the application interface to the spell checker.

Input Parameters

None

Return Value

None

Special Notes

spell_bind_end should be called at the end of an application in which **spell_bind_init** was called.

DeskMate Technical Reference

Desk Executive

int **spell_bind_init** ()

spell_bind_init initializes a spell checker session.

Input Parameters

None.

Return Value

DM_ERROR if SPELL.RES or SPL.RES cannot be loaded.

NULL if successfully loaded.

Special Notes

spell_bind_init loads the Spell Checker Engine (spl.res), loads and initializes the user dictionary, and initializes the internal cache.

Any application which uses any of the spell checker functions must first initialize the spell checker with **spell_bind_init**.

DeskMate Technical Reference
Desk Executive

Thesaurus Resource Initialization Calls

thes_bind_end ()

thes_bind_end terminates the applications interface to the thesaurus.

Input Parameters

None.

Return Value

None.

Special Notes

thes_bind_end should be called at the end of an application if thes_bind_init was called.

DeskMate Technical Reference

Desk Executive

thes_bind_init ()

thes_bind_init initializes a thesaurus session.

Input Parameters

None.

Return Value

DM_ERROR if THES.RES cannot be loaded.

NULL if THES.RES is successfully loaded.

Special Notes

Any application which wishes to use the thesaurus functions must first initialize the thesaurus with **thes_bind_init**.

DeskMate Development System
Technical Reference
03.05.00

"Dialog Box Manager"

Dialog Box Manager

Table of Contents

General Description/Notes	8- 1
Structures/Defines	8- 2
dlg_drawDisplays a dialog box.....	8- 7
dlg_draw_frameDraws a dialog box frame.....	8- 8
dlg_runRuns a dialog box.....	8- 9

General Description/Notes

The Dialog Box Manager allows the application to prompt the user for several pieces of information simultaneously. It manages all of the components within the dialog box and returns to the application when an action has taken place on a component.

Control is passed back to the application only if an action is taken on a component in the dialog box. A change of focus without a change of state in a component will **NOT** return control to the application.

The coordinates of the component structures within a dialog box are placed relative to the dialog box window and are in world coordinate terms. The dialog box itself is relative to the base window.

All components have a return code and upon exit from a dialog box, the return code of the most recent component action is returned and also is placed in the dialog box structure. Return codes for Dialog Boxes and return codes for Menu Bars should always be unique. For example define dialog box return codes to be `APP_TAG + 1`, `APP_TAG + 2`, `APP_TAG + 3`, etc, and menu bar return codes to be `APP_TAG + 100`, `APP_TAG + 101`, `APP_TAG + 102`, etc.

Buttons all have an on/off state. They are toggled by a mouse click on them or by hitting the space bar when the button has the focus.

The tab key changes focus to the next component as determined by the order of the application assigned focus order in the component pointer list of the dialog box, and the "back-tab" key changes focus to the previous component.

Upon entry into the dialog box, all push buttons should be in the `BUTTON_UP` state.

In most cases, each dialog box should have the standard OK and CANCEL push buttons. This creates a standard and will always allow a path back to the application. The application must define these push buttons. The OK button should have an accelerator of `DLG_OK_KEY` and the CANCEL button should have an accelerator of `DLG_CANCEL_KEY` (defined in `CSR_CMPS.H`). In general, no other components in a dialog box should have these accelerators. Static strings and boxes in dialog boxes are not interactive.

When using a menu bar in a dialog box, the menu bar must **not** be in the tab order. To make any component not be accessible through the tab and cursor keys set `bEnabled` to `CMP_NO_TAB`.

DeskMate Technical Reference

Dialog Box Manager

Structures/Defines

```
/*-----*/
/*
/* CSRCMPS.H contains the Dialog Box structures and defines */
/*
/*-----*/
```

Dialog boxes and information boxes use the titled frame structure. A frame defines the outer boundary of a dialog or information box.

```
/* Titled frame */
struct frame_defn
{
    MAPRECT      maprect; /* origin/extent of the frame */
    unsigned char *pString; /* pointer to title string for the frame */
};
typedef struct frame_defn FRAME;
```

Dialog boxes use static strings as labels and messages.

MAPRECT:

Defines the location and size of the frame. The MAPRECT structure is defined in CSRBASE.H and should be included in the application.

pString:

The pointer to the title string to be displayed in the frame.

```
/* Static string */
struct static_string_defn
{
    char          bEnabled; /* grayed flag */
    int           xorg;     /* x origin of string */
    int           yorg;     /* y origin of string */
    char          attr;     /* string attribute */
    unsigned char *pString; /* pointer to string */
};
typedef struct static_string_defn STATIC_STRING;
```

bEnabled:

This is the state the string is drawn in. If bEnabled is DISABLED, the string is drawn in the grayed state. If bEnabled is ENABLED, the string is drawn in the normal state. Static strings can be used as labels for other components. Use the grayed state when the component described by the string cannot be selected in the application.

xorg and yorg:

The x and y screen coordinates of the first character of the string.

attr:

The text attributes of the characters in the string. Two attributes are supported, BOLD and UNDERLINE. These are defined in CSRVID.H

pString:

A pointer to the string definition in the application's data area.

Static boxes visually separate similar or related component groups within dialog boxes. For example, a radio button group is usually surrounded by a static box to separate it from other components in a dialog box.

DeskMate Technical Reference Dialog Box Manager

```
/* Static box */
struct static_box_defn
{
    char    type;          /* type of box */
    MAPRECT maprect;       /* org/ext of box */
    char    color;         /* color of the box */
};
typedef struct static_box_defn STATIC_BOX;
```

type:

Describes the kind of box to be used. The following identify the acceptable values for the type element in the STATIC_BOX structure:

```
/* Box types */
FLAT_BOX      0          /* single thin line box */
RAISED_BOX   1          /* 3-D raised block */
PYRAMID       2          /* 3-D pyramid */
DEST_FLAT_BOX 3          /* thin line box that erases a RAISED_BOX */
LISTBOX_BOX   4          /* item list border for list boxes */
GRAYED_BOX    0x80       /* flag to make boxes grayed */
ENLARGED_BOX  0x40       /* flag to pad frame interior with gap */
```

MAPRECT:

Defines the location and size of the box. The MAPRECT structure is defined in CSRBASE.H and should be included in the application.

color:

Selects a valid color (COLOR1 to COLOR16) for the interior of the box.

No. color is the color of the box frame. The interior is always the current background color.

Static icons are line drawings used to describe options, states, etc.

```
/* Static Icon */
struct static_icon_defn
{
    MAPRECT maprect;       /* icon org/ext */
    char    *pIcon;        /* pointer to icon definition */
};
typedef struct static_icon_defn STATIC_ICON;
```

MAPRECT:

The location and size of the icon. The MAPRECT structure is defined in CSRBASE.H and should be included in the application.

pIcon:

The pointer to the icon definition.

The elements in a dialog box structure define the appearance of the dialog box and some aspects of how the box will be used.

```
/* Dialog box */
struct dialog_box_defn
{
    char          bStatRedraw;    /* redraw flag for static items */
    unsigned int  return_value;   /* return code */
    char          focus_index;    /* next focus id */
    char          help_level;     /* help level */
    FRAME         *pFrame;        /* pointer to FRAME structure */
    char          nStrings;       /* number of static strings */
    STATIC_STRING *pStrings;      /* ptr to STATIC STRING structs */
    char          nBoxes;         /* number of static boxes */
};
```

DeskMate Technical Reference

Dialog Box Manager

```

STATIC_BOX      *pBoxes;          /* pointer to STATIC BOX structs */
char            nIcons;           /* number of static icons */
STATIC_ICON     *pIcons;          /* pointer to STATIC ICON structs */
char            nCmps;            /* number of components */
char            *pRFlags;         /* ptr to component redraw flags */
char            **pCmps;          /* ptr to component pointer list */
};
typedef struct dialog_box_defn DIALOG_BOX;

```

DIALOG_BOX.bStatRedraw:

Redraw flag for static items of the dialog box (DLG_REDRAW or DLG_NO_REDRAW). See the pRFlags element below to redraw components.

DIALOG_BOX.return_value:

The return code returned by the component with the last action.

DIALOG_BOX.focus_index:

The index of the component with the current focus, used to get the focus the next time the box is run. An index of zero refers to the first component in the component list (pCmps).

DIALOG_BOX.help_level:

HELP_NULL is the only valid value for this element.

DIALOG_BOX.pFrame:

Pointer to FRAME structure

FRAME.maprect:

MAPRECT.xorg	world coordinate x origin of the area to frame.
MAPRECT.yorg	world coordinate y origin of the area to frame.
MAPRECT.xext	world unit x extent of the area to frame.
MAPRECT.yext	world unit y extent of the area to frame.
FRAME.pString	pointer to the title string of the frame.

DIALOG_BOX.nStrings:

Number of static strings in the dialog box.

DIALOG_BOX.pStrings:

Pointer to the STATIC_STRING structure for the dialog box.

STATIC_STRING.xorg	world coordinate x origin of string.
STATIC_STRING.yorg	world coordinate y origin of string.
STATIC_STRING.attr	character attribute to display the string with. Any valid character attribute may be used.
STATIC_STRING.pString	pointer to the string.

DIALOG_BOX.nBoxes:

The number of static boxes in the dialog box.

DIALOG_BOX.pBoxes:

Pointer to the STATIC_BOX structures.

STATIC_BOX.type	type of frame to display as defined below.
/* Box types */	
FLAT_BOX	0

DeskMate Technical Reference

Dialog Box Manager

RAISED BOX	1	
PYRAMID	2	
DEST FLAT BOX	3	
LISTBOX BOX	4	
GRAYED BOX	0x80	
ENLARGED BOX	0x40	

STATIC_BOX.maprect: —

MAPRECT.xorg	world coordinate x origin relative to the origin of the dialog box of the area to draw the frame around.
MAPRECT.yorg	world coordinate y origin relative to the origin the dialog box of the area to draw the frame around.
MAPRECT.xext	world unit x extent of the area to draw the frame around.
MAPRECT.yext	world unit y extent of the area to draw the frame around.
STATIC_BOX.color	color of the frame (COLOR1 to COLOR16).

DIALOG_BOX.nIcons:

The number of static icons in the dialog box.

DIALOG_BOX.pIcons:

Pointer to the STATIC ICON structures.

STATIC_ICON.maprect:	
MAPRECT.xorg	world coordinate x origin of the icon relative to the origin of the dialog box.
MAPRECT.yorg	world coordinate y origin of the icon relative to the origin of the dialog box.
MAPRECT.xext	world unit x extent of the icon.
MAPRECT.yext	world unit y extent of the icon.
STATIC_ICON.pIcon	pointer to a graphic list defining the icon.

DIALOG_BOX.nCmps:

Number of components in the dialog box.

DIALOG_BOX.pRFlags:

Pointer to an array of component redraw flags. There must be a 'char' for each component in the dialog box. Use DLG_REDRAW or DLG_NO_REDRAW for each redraw flag.

DIALOG_BOX.pCmps:

A pointer to the component pointer list. The order of the component entries in this list determines the "TAB order" of the components in the dialog box. The TAB key will move the focus "down" the list and the BACKTAB key will move the focus "up" the list.

Frame types:

```

/* Box types */
FLAT BOX      0      /* single thin line box */
RAISED BOX    1      /* 3-D raised block */
PYRAMID       2      /* 3-D pyramid */
DEST FLAT BOX 3      /* thin line box that erases a RAISED BOX */
LISTBOX BOX   4      /* item list border for list boxes */
GRAYED BOX    0x80   /* flag to make boxes grayed */
ENLARGED_BOX  0x40   /* flag to pad frame interior with gap */

/* Static item redraw flags */
DLG_REDRAW      SELECTED  /* causes dlg_run to redraw the static */
                                /* string, boxes and icons in the */
                                /* dialog box. */
DLG_NO_REDRAW    DESELECTED /* prevents dlg_run from re-drawing the */

```

DeskMate Technical Reference

Dialog Box Manager

```
/* static strings, boxes and icons in */
/* the dialog box. */

/* No dialog box help */
DLG_NO_HELP      CSR_NULL

/* OK and Cancel button accelerators */
DLG_OK_KEY      RETURN_KEY /* the accelerator for the 'OK' button */
/* in a dialog box */
DLG_CANCEL_KEY  ESC_KEY    /* the accelerator for the 'Cancel' */
/* button in a dialog box */

/* Request for event appl to be returned to application */
/* or in help level */
DLG_APPL_REQUEST (MENU_TAG+01) /* return code for */
/* appl events */
DLG_RETURN_APPLS 0x80          /* all appl events to */
/* be returned */

/* keep a component out of tab order */
/* to be ORed into the bEnable element */
/* of the specific component that needs */
/* to be out of the tab order */
CMP_NO_TAB      0x02
```

DeskMate Technical Reference Dialog Box Manager

dlg_draw (pDIALOG_BOX)
DIALOG_BOX *pDIALOG_BOX;

dlg_draw is used to display a dialog box.

Input Parameters

pDIALOG_BOX is a pointer to a dialog box structure.

Return Value

None.

Special Notes

Dialog boxes are always drawn relative to the base window origin.

Use the following formulas to determine the origin/extent of the area of the screen to preserve for a dialog box:

```
xorg = FRAME.xorg-CHAR_XEXT  
yorg = FRAME.yorg-((CHAR_YEXT*3)/2)  
xext = FRAME.xext+(2*CHAR_XEXT)  
yext = FRAME.yext+(2*CHAR_YEXT)
```

DeskMate Technical Reference

Dialog Box Manager

dlg_draw_frame (pFRAME)
FRAME *pFRAME;

dlg_draw_frame draws the frame with the title in top of the frame.

Input Parameters

pFRAME is a pointer to a FRAME structure.

Return Value

None.

Special Notes

Use the following formulas to determine the origin/extent of the area of the screen to preserve for a dialog box:

```
xorg = FRAME.xorg-CHAR_XEXT  
yorg = FRAME.yorg-((CHAR_YEXT*3)/2)  
xext = FRAME.xext+(2*CHAR_XEXT)  
yext = FRAME.yext+(2*CHAR_YEXT)
```

DeskMate Technical Reference

Dialog Box Manager

dlg_run (pDIALOG_BOX)
DIALOG_BOX *pDIALOG_BOX;

dlg_run runs the dialog box.

Input Parameters

pDIALOG_BOX is a pointer to a dialog box structure.

Return Value

The return code of the component with the last action.

Special Notes

dlg_run is used to get the next update of the dialog box. The **bStatRedraw** flag in the dialog box is only used to force the dialog box to redraw the Static component(s) if there is any Static component(s) which have changed.

The **pRFlags** pointer is a pointer to a redraw flag list. Each element of the list is a flag corresponding to a component that indicates whether or not that component is to be redrawn.

There are two areas that the applications may use for re-drawing dialog box pieces. The redraw structure element, **DIALOG_BOX.bStatRedraw**, is used to redraw all static icons, strings, and boxes. The second structure element, **DIALOG_BOX.pRFlags**, is a pointer to an array of characters that number exactly the same as the number of components. Each of these chars may be set to **DLG_REDRAW** or **DLG_NO_REDRAW**. The **dlg_run** call will selectively redraw the components you request. This allows you to optimize your code to only redraw the component(s) that have been changed by the application. Remember that components are displayed in their correct current state when the **dlg_run** call returns to the application. A good example is the listbox/edit field type of dialog box. As the listbox changes you see from **DIALOG_BOX.return_value** that a **CMP_SELECT_CHANGE** has occurred. Simply update the edit field structure, set the edit field redraw flag to **DLG_REDRAW**, and call **dlg_run**. Only the edit field will be re-displayed. After **dlg_run** re-draws it will automatically reset the flag to **DLG_NO_REDRAW**.

For dialog boxes, the return value of **dlg_run** will continue to return the **CMP_HEADER.return_code** of the component that an action has occurred in. The type of action (ie., the return value from the actual component) will be placed in the **DIALOG_BOX.return_value**.

This value may be any of the **CMP_** returns defined in the component documentation. If the value is **CMP_SELECT_CHANGE**, the component is informing the application that a change in its' select status has occurred. A change in select status may mean the user has begun selecting text or terminated a selection in an edit field, or the user has moved the arrow key on a single selection listbox (thus deselecting the current item and selecting the next item). NO EVENT IS LEFT ON THE EVENT QUEUE TO BE READ. Except for mouse hold on Push buttons. Therefore, if other events are not required, and **event_purge** should be called to stop mouse hold events being returned and the pointer being fenced within the push button component.

For the **CMP_ACTION** return, the component has had a change in its' data and focus is now moving to another component. The application may see the component return value in the dialog box structure. The application must either return to the dialog box manager via **dlg_run** without

See cmp-run() for codes.

DeskMate Technical Reference

Dialog Box Manager

clearing the event queue, or terminate the dialog box and clear the event queue via **event_purge**.

Environment Manager

"Environment Manager"

Table of Contents

General Description/Notes	9- 1
Structures/Defines	9- 2
env_addAppends specified to existing data.....	9- 3
env_closeRemoves configuration data from memory.....	9- 4
env_deleteRemoves all configuration data.....	9- 5
env_getSetup configuration data for use.....	9- 6
env_openOpens and loads configuration data.....	9- 7
env_replaceReplaces configuration data on disk.....	9- 8
env_writeWrites configuration data to disk.....	9- 9

DeskMate Technical Reference

Environment Manager

General Description/Notes

DeskMate applications which need to store application specific configuration information on the disk and retrieve it when the application is run may do so through the Environment manager. Accessing configuration files involves file opens, reads, writes, disk swapping, user interface, and error processing that is not easily duplicated in any application that requires configuration information.

The environment manager stores configuration information for any DeskMate application, and handles all disk I/O and user interface in a consistent manner that does not cost the application considerable code space.

The environment manager provides two major benefits to DeskMate applications:

1. All configuration information is easily available to any other DeskMate application or accessory through a standard application interface.
2. Each application and accessory that needs configuration information does not have to perform the complicated file I/O and disk swapping to store and access the information.

The environment manager is similar in concept to the MS-DOS environment specified with "set" commands by the computer user. Unlike the MS-DOS environment, which only stores null-terminated ASCII strings, DeskMate applications may store both ASCII and binary configuration information. Another special feature is that the environment manager will manage multiple environment information files simultaneously.

The application that uses the environment manager requests that it be loaded at initialization with the call to **guf_bind_init**. The application issues the **env_open** call, specifying the name and location of the environment information file that contains the environment information it needs access to. The environment manager loads the file, and the application may then access and update information through calls to **env_get**, **env_add**, **env_replace** and **env_delete**. When it is necessary for the application to have the environment information updated to the disk, the application does so with a call to **env_write**.

The environment manager maintains all information in memory while DeskMate is active, and does not read or write environment files unless specified by an application. The information may be removed from memory by a call to **env_close**. When reading and writing environment files, the application may cause the environment manager to look only on the current disk for the environment file, or may optionally cause the environment manager to perform disk swapping until the user inserts the disk containing the file or the user cancels.

DeskMate Technical Reference Environment Manager

Structures and Defines

All calls to the environment manager require a pointer to an environment data information structure. This structure and these defines may be found in dmguif.h and dmguif.inc.

```
/*-----*/
/*
/* DMGUF.H contains the Environment structures and defines */
/*
/*-----*/

typedef struct env_defn
{
    char *pEnvFileName; /* environment data file being accessed */
    char *pDosEnvString; /* MS-DOS environment variable */
    char bSwap;          /* specifying path of data file */
    char *pDmEnvString;  /* flag to disk swap until disk is */
    char far *pDataInfo; /* found containing env data file */
    int DataLen;         /* name of DeskMate env data being */
    /* accessed in data file */
    /* pointer to the specified */
    /* environment data */
    /* length in bytes of the environment data */
} ENVDATA;
```

Values for bSwap are:

ENV_SWAP_DISABLED	0	/* look on the current disk only for */ /* the file */
ENV_SWAP_ENABLED	1	/* if file is not found, prompt user */ /* to insert disk and retry */
ENV_SWAP_CREATE	2	
ENV_NO_CREATE	3	/* allows an application to create */ /* temporary memory-only environment */ /* data which is only used during a */ /* single DeskMate session */
ENV_CANCEL	-2	

DeskMate Technical Reference Environment Manager

env_add (pENVDATA)
ENVDATA *pENVDATA;

env_add *appends* the specified configuration data to the end of the existing configuration data that is referred to by the name specified by pDmEnvString. If no configuration data exists for pDmEnvString, then a new entry for the configuration information will be started.

Input Parameters

pEnvFileName must point to a string specifying the name of the environment data file that the application wishes to update. This string may be a file name or a complete pathname.

pDmEnvString must point to a string specifying the name of the DeskMate environment variable that references the environment data that the application wishes to update.

pData points to the first byte of the application's environment data to be added to the specified environment data file. The environment manager copies this information from the application into its own data buffer.

DataLen specifies the number of bytes of environment data that the application wishes to add to the environment data file.

Return Values

<int>

CSR_ERROR if there is not enough memory in the environment manager to store the specified environment data, or the filename has not been previously opened.

DeskMate Technical Reference Environment Manager

env_close (pEnvData)
ENVDATA *pEnvData;

env_close removes all configuration data from memory for the specified file.

Input Parameters

pEnvFileName must point to a string specifying the name of the environment data file that the application wishes to access. This string may be a file name or a complete pathname.

pDosEnvString must point to a string specifying the name of the MS-DOS environment variable that specifies the path of where to find the environment data file. If no variable exists in the MS-DOS environment under this name, then the environment manager will access the file in the subdirectory specified by the pEnvFileName string. If pEnvFileName does not specify a drive and directory, then the environment manager will look for the data file in the current directory. pDosEnvString cannot contain multiple paths (ie: the PATH environment variable is not allowed).

Return Value

CSR_ERROR if the specified data is not in memory.

Special Notes

env_close WILL NOT write any data to disk before removing it from memory. **env_close** must be called with the same pEnvFileName and pDosEnvString as when it was opened.

DeskMate Technical Reference

Environment Manager

env_delete (pEnvData)
ENVDATA *pEnvData;

env_delete removes configuration data referred to by the name specified by pDmEnvString from the specified environment data file.

Input Parameters

pEnvFileName must point to a string specifying the name of the environment data file that the application wishes to update. This string may be a file name or a complete pathname.

pDmEnvString must point to a string specifying the name of the DeskMate environment variable that references the environment data that the application wishes to delete.

Return Values

<int>

CSR_ERROR if no environment data exists under the name specified by pDmEnvString, or the filename has not been previously opened.

Special Notes

The file will remain registered in memory and if it has multiple pDmEnvString's, the other strings will remain. The disk file will not be updated by **env_delete**.

DeskMate Technical Reference Environment Manager

env_get (pEnvData)
ENVDATA *pEnvData;

env_get sets the far pointer, pData to the first byte of configuration data that is referred to by the name specified by pDmEnvString.

Input Parameters

pEnvFileName must point to a string specifying the name of the environment data file that the application wishes to access. This string may be a file name or a complete pathname.

pDmEnvString must point to a string specifying the name of the DeskMate environment variable that references the environment data that the application wishes to access.

pData points to the first byte of data in the environment manager data buffer that is referenced by the specified DeskMate environment variable in the specified environment data file. The application should copy this information from the environment manager into it's own data buffer.

DataLen specifies the number of bytes of environment data for pDmEnvString.

Return Values

<int>

CSR_ERROR if no environment data exists under the name specified by pDmEnvString, or the filename has not been previously opened.

DeskMate Technical Reference Environment Manager

env_open (pEnvData)
ENVDATA *pEnvData;

env_open opens the configuration data file specified by pEnvFileName and loads it into memory.

Input Parameters

pEnvFileName must point to a string specifying the name of the environment data file that the application wishes to access. This string may be a file name or a complete pathname.

pDosEnvString must point to a string specifying the name of the MS-DOS environment variable that specifies the path of where to find the environment data file. If no variable exists in the MS-DOS environment under this name, then the environment manager will access the file in the subdirectory specified by the pEnvFileName string. If pEnvFileName does not specify a pathname, then the environment manager will look for the data file in the current directory. pDosEnvString cannot contain multiple paths (ie: the PATH environment variable is not allowed).

bSwap specifies whether to disk swap to find the environment data file. If set to ENV_SWAP_ENABLED it will look for the file in the path specified by pDosEnvString and pEnvFileName. If the file is not found, it calls **dm_file_search** to determine if it exists anywhere in the system. **dm_file_search** either finds the file or prompts the user to insert the disk containing the file with RETRY and CANCEL prompts. If the user CANCEL's, the file is not created. If bSwap is set to ENV_SWAP_DISABLED it looks on the current disk only, if not found it creates the empty file. If bSwap is set to ENV_NO_CREATE and if the file is not already opened and registered with the environment manager it registers the file in RAM. It does **NOT** do any searching for the file, and does **NOT** create the file.

Return Values

<int>

CSR_ERROR if:

- an error occurred opening or loading the data file.
- an error occurred creating the data file.
- there is not enough environment memory left to load the data file.

ENV_CANCEL if:

- the user chose to CANCEL from disk swapping.
- if bSwap is ENABLED the file did not get created.

None of the above is correct, at least if pDosEnvString is "DMCONFIG".

If pDosEnvString is a null string, the environment manager will be corrupted, causing lockup under DeskMate 3.02. (?) Under DM302, pEnvFileName cannot be a complete path—it must be in the current directory, or the directory specified by the

pDosEnvString variable. Set pDosEnvString to NULL and use Set_Directory() to load a specific environment file in a specific directory.

DeskMate Technical Reference Environment Manager

env_replace (pEnvData)
ENVDATA *pEnvData;

env_replace *replaces* the specified configuration that is referred to by the name specified by pDmEnvString. If no configuration data exists for pDmEnvString, then a new entry for the configuration information will be started.

Input Parameters

pEnvFileName must point to a string specifying the name of the environment data file that the application wishes to update. This string may be a file name or a complete pathname.

pDmEnvString must point to a string specifying the name of the DeskMate environment variable that references the environment data that the application wishes to update.

pData points to the first byte of the application's environment data to be added to the specified environment data file. The environment manager copies this information from the application into its own data buffer.

DataLen specifies the number of bytes of environment data that the application wishes to add to the environment data file.

Return Values

<int>

CSR_ERROR is returned if there is not enough memory in the environment manager to store the specified environment data, or the filename has not been previously opened.

DeskMate Technical Reference Environment Manager

env_write (pEnvData)
ENVDATA *pEnvData;

env_write writes all current configuration data in the specified environment data file to the disk.

Input Parameters

pEnvFileName must point to a string specifying the name of the environment data file that the application wishes to write. This string may be a file name or a complete pathname.

pDosEnvString must point to a string specifying the name of the MS-DOS environment variable that specifies the path of where to find the environment data file. If no variable exists in the MS-DOS environment under this name, then the environment manager will write the file in the subdirectory specified by the pEnvFileName string. If pEnvFileName does not specify a pathname, then the environment manager will write the data file in the current directory only.

If bSwap is set to ENV_SWAP_ENABLED or ENV_SWAP_CREATE, it searches for the file via the pDosEnvString and pEnvFileName. If the file is not found, it calls **dm_file_search**. If **dm_file_search** cannot find the file, the user will be prompted to insert a disk containing the file or CANCEL. If the file is found, the data will be written to disk. If the user CANCEL's and bSwap is set to ENV_SWAP_ENABLED, the file is not created. If the user CANCEL's and **bSwap** is set to ENV_SWAP_CREATE, the file is created and written. If bSwap is set to ENV_SWAP_DISABLED it looks on the current disk only, if not found, it creates the file and writes it.

Return Values

<int>

CSR_ERROR if an error occurred opening or writing the data file or there is not enough disk space to write the data file.

ENV_CANCEL if the user chose to CANCEL from disk swapping

Event Manager

"Event Manager"

Table of Contents

General Description/Notes	10- 1
Structures/Defines	10- 3
event_add_edriverAdds a user defined event driver.....	10- 6
event_add_interpAdds a user defined interpreter.....	10- 7
event_delete_edriver .Deletes a user defined event driver.....	10- 8
event_delete_interp ..Deletes a user defined interpreter.....	10- 9
event_get_uninterp ...Gets an uninterpreted event.....	10-10
event_purgePurges event from event queue.....	10-11
event_readReturns an event.....	10-12
event_scanScans for an event.....	10-13
event_writeWrites an event to the event queue.....	10-14

DeskMate Technical Reference Event Manager

General Description/Notes

The event manager will return events from both the keyboard and the pointing device. All events returned have a message type, a parameter, and a set of coordinates for mouse events. The event manager will also determine if the event has occurred inside or outside the currently active window. If an event is outside the active window, an `EVENT_OUTSIDE` message will be returned.

There is a maximum of two events stored at any one time.

About Hot Spots:

A hot spot is an item or an area of the display that has been defined by the application as being an item on display area that it wishes the event manager to monitor. When an action occurs on a hot spot, the event manager notifies the application by returning a command event (see below). Hot spots may only be defined by opening a component or drawing a main menu bar. Generally, a hot spot will have a keyboard accelerator and/or a display rectangle defined for it. If the user presses the keyboard accelerator or generates a mouse button event while the mouse pointer is within the defined display rectangle, the event manager will translate the event to the appropriate command event. Only the window in which the hot spot was created may receive the command event for that hot spot. All other windows will be returned as an outside event, with the exception of mouse events on covered hot spots, as described below for `EVENT_COMMAND`.

There is a fixed 2K buffer used to track windows and hot spots. Each newly created window uses 40 hex bytes, each hot spot uses 12 hex bytes. For example:

MAX window buffer size	=	2048 bytes (39 windows)
each win_group_init	=	- 68 bytes (done by each csr_init)

		1980 bytes
menu bar entry	=	- 16 bytes (hot spot entry)

		1964 bytes
each menu button	=	x bytes (No. buttons * 32)

		BUFFER
each menu item w/accel	=	x bytes (No. items * 16)

		BUFFER

For each menu button F9 or F10 it takes one hot spot. (16 bytes each)

Each window created by an application (win_open) is one window. (52 bytes)

Each component created/opened (cmp_open (non push button)) is one hot spot. (16 bytes)

Each push button with `DEFAULT_ACCELERATOR` is 2 hot spots. (32 bytes)

Each push button with other than `DEFAULT_ACCELERATOR` is one hot spot. (16 bytes)

$$\text{no. windows available} = \frac{\text{BUFFER} - ((\text{num windows} * 52) + (\text{num hot spots} * 16))}{52}$$
$$\text{no. hot spots available} = \frac{\text{BUFFER} - ((\text{num windows} * 52) + (\text{num hotspots} * 16))}{16}$$

Note: You must allow for one window entry to exist (free), to use the component manager.

DeskMate Technical Reference

Event Manager

There are currently six types of events that may be returned:

EVENT_COMMAND: A command event is returned when an event caused by the mouse click or pressing an accelerator of another component in the *current* active window. A command event is *not* returned when the focus is on the component and either the mouse click or an accelerator key is pressed on that component.

EVENT_CHAR: A character event is a keyboard action and the key code is returned in the event parameter when this message type is returned.

EVENT_MOUSE: The return of a mouse event indicates that a mouse click has occurred. The type of the mouse click can be found in the event parameter and the position of the mouse pointer can be found in the event x and y elements. Note that if the mouse is clicked on a component, the action will be returned as an **EVENT_COMMAND** and not an **EVENT_MOUSE**. When a mouse button hold sequence is initiated, the mouse pointer will be limited to the active window until the button is released, and therefore, a hold sequence cannot generate an outside event. Also, only the window that read the mouse button down event will be able to read any other events associated to that hold sequence.

EVENT_OUTSIDE: An outside message type is returned when an action has occurred outside the currently active window. The application must decide upon receipt of an outside event whether to activate the parent window to process the event or to ignore the outside event. If the application chooses to process the event, it must activate the parent window and call **event_read** again to get the event message type and parameter. If the application chooses to ignore the event, it must purge the pending event with **event_purge** before making any other calls to **event_read**.

EVENT_APPL: This message type indicates that a special CSR defined event has occurred. There are six types of parameters that may accompany this event: **APPL_ACCESS**, **APPL_REDRAW_VID**, **APPL_REDRAW_VGM**, **APPL_TASK_SWITCH**, **APPL_STAT**, and **APPL_ALARM**. If **APPL_ACCESS** is returned, it means that an accessory has been chosen by the user. It is the responsibility of the application to run the accessory through an Executive routine. The number of the accessory to run can be found in the event x element. If **APPL_REDRAW_VID** is returned, it means that the application must redraw its display. If **APPL_REDRAW_VGM** is returned, it means the application needs to redraw its form. If **APPL_TASK_SWITCH** is returned it means the user has requested a Task Switch. **APPL_STAT** is used for the applications interpreters, explained below. An **APPL_ALARM** is returned when an alarm item has been selected.

EVENT_NULL: This message type is only returned by the **event_scan** routine and indicates that there is no event available.

About key codes returned by the Event Manager:

All normal ASCII key codes are returned in an **EVENT** structure in the application's data area. If the user strikes a normal ASCII key, the "msg" element of the **EVENT** structure will be **EVENT_CHAR** and the "param" element will hold the ASCII code of the key.

Other non-ASCII keys are returned in the same way, and the application should use the CSR key defines to reference these keys. It is suggested that the keys be referenced by their functions described in the Keyboard Manager section of this manual.

DeskMate Technical Reference

Event Manager

Structures/Defines

```

/*-----*/
/*
/* CSRBASE.H contains the Event structure and defines */
/*-----*/

/* Event Manager */
struct event_defn
{
    char        msg;           /* event type */
    unsigned int param;        /* event parameter */
    int         x;             /* event mouse x */
    int         y;             /* event mouse y */
};
typedef struct event_defn EVENT;

EVENT.msg      char    - event type as defined below.
EVENT.param    int     - event parameter as defined below.
EVENT.x        int     - normalized world coordinate x position of the
                        event (-32768 to 32768). This element is
                        valid only if 'msg' is EVENT_MOUSE.
EVENT.y        int     - normalized world coordinate y position of the
                        event (-32768 to 32768). This element is
                        valid only if 'msg' is EVENT_MOUSE.

/* Event types for msg element */
EVENT_NULL      0      /* no event only returned by event scan */
EVENT_CHAR      1      /* character event generated by */
EVENT_MOUSE     2      /* keyboard (destructive) */
EVENT_MOUSE     2      /* mouse click event - generated by */
EVENT_COMMAND   3      /* pointing device (destructive) */
EVENT_COMMAND   3      /* command event - event on a hotspot */
EVENT_OUTSIDE   4      /* (destructive) */
EVENT_OUTSIDE   4      /* outside event - event outside current*/
EVENT_OUTSIDE   4      /* active window (non-destructive) */
EVENT_REEVALUATE 5      /* reevaluate raw event - forces a */
EVENT_REEVALUATE 5      /* reevaluation of the raw event which */
EVENT_REEVALUATE 5      /* generated the most recent event. */
EVENT_REEVALUATE 5      /* This event type will never be */
EVENT_REEVALUATE 5      /* returned from the Event Manager, but */
EVENT_REEVALUATE 5      /* is used when using 'event write' */
EVENT_APPL      6      /* special application message */
EVENT_MENU      7      /* (destructive) */
EVENT_MENU      7      /* special menubar event */

/* Mouse event types for param element */
MS_CLICK        (MOUSE_TAG+1) /* single click */
MS_DBL_CLICK    (MOUSE_TAG+2) /* double click */
MS_BUTTON_DOWN  (MOUSE_TAG+3) /* button down (begin repeat) */
MS_HOLD         (MOUSE_TAG+4) /* button held (repeating) */
MS_BUTTON_UP    (MOUSE_TAG+5) /* button released (end repeat) */
MS_SHIFT_CLICK  (MOUSE_TAG+6) /* shifted click */

```

DeskMate Technical Reference

Event Manager

```

/* Application messages for param element */
APPL REDRAW VID      1      /* redraw video */
APPL REDRAW VGM      2      /* redraw form */
APPL_ACCESS          3      /* an accessory has been selected by */
                          /* the user that the application needs */
                          /* to invoke. */
APPL_TASK_SWITCH     4      /* task switch requested */
APPL_STAT            5      /* request for immediate Appl attention */
APPL_ALARM           6      /* an alarm item has been selected */

/* Definable event drivers and interpreters */
struct edriver_defn
{
    int      (far *pInit)();    /* This element is a far pointer to the
                                driver initialization procedure.
                                The initialization procedures are
                                called when the CSR received a new
                                task data segment. This routine is
                                responsible for initializing any
                                task data within the module that it
                                resides that is pertinent to the
                                event driver.
                                ENTRY: None.
                                EXIT: None.*/

    int      (far *pRead)();    /* This element is a far pointer to the
                                driver read procedure. This routine
                                is called by event scan/event read
                                only when none of the built-in event
                                drivers and none of the event
                                drivers registered prior to this
                                event driver have no events pending.
                                This procedure typically should
                                return the pending event and remove
                                the event from its own queue (if
                                any). If the event driver has no
                                event pending, it must return a null
                                event.
                                ENTRY: ds:di points to an EVENT
                                structure
                                EXIT: the passed event structure
                                must contain a valid event of the
                                event driver or a null event. */

    int      (far *pResrv1)();  /* This element is reserved for future
                                use of event drivers. This should
                                point to a far return. */

    int      (far *pResrv2)();  /* This element is reserved for future
                                use of event drivers. This should
                                point to a far return. */
};

typedef struct edriver_defn EDRIIVER;

struct interp_defn
{
    char      priority;         /* This element specifies the
                                priority of the event interpreter.
                                If it is set to EVENT HIGH PRIORITY,
                                the interpreter will be called
                                before the built-in interpreters.
                                If it is set to EVENT LOW PRIORITY,
                                the interpreter will be called after
                                the built-in interpreters. No
                                interpreters are allowed to take
                                precedence over the record/playback
                                interpreter.*/

```

DeskMate Technical Reference

Event Manager

```
int    (far *pInterp) ();    /* This element is a far pointer to
                               interpreter procedure. This routine
                               must preserve all registers and the
                               direction flag. This procedure may
                               or may not modify the current event
                               by overwriting the current event.
                               The interpreter may not cause the
                               event manager to "drop" events.
                               Events may be modified, but should
                               never be lost. In particular, a
                               mouse button up event must pass
                               through the built-in interpreters
                               for proper mouse hold processing to
                               occur.
                               ENTRY: ds:di = pointer to an EVENT
                               structure containing the current
                               event.
                               EXIT:  None.*/

};
typedef struct interp_defn INTERP;

EVENT_HIGH_PRIORITY      1
EVENT_LOW_PRIORITY       2
```

DeskMate Technical Reference Event Manager

event_add_edriver (pEDRIVER)
EDRIVER *pEDRIVER;

event_add_edriver allows a CSR user defined device driver to be polled by the event manager.

Input Parameters

pEDRIVER is a pointer to an EDRIVER structure.

Return Value

<int>

The handle of the event driver.

CSR_ERROR if the device driver table is full.

Special Notes

The event driver's structure is added to the event manager's driver list and will be considered to be available at all times. The module in which the driver routines reside must remove the event driver if it terminates without remaining resident. All of the driver routines specified by the EDRIVER structure must point to executable routines, i.e. if no routine is required for a procedure then the pointer must point to a far return.

The added event driver routines must preserve the ES segment register.

The added event driver should not make direct calls into DOS.

The added event drivers have a lower priority than the mouse and keyboard drivers. The event drivers initialization routine will be called when added.

See Also

EDRIVER structure listed above.

DeskMate Technical Reference

Event Manager

event_add_interp (pINTERP)
INTERP *pINTERP;

event_add_interp adds a CSR user event interpreter to the event manager.

Input Parameters

pINTERP is a pointer to an INTERP structure.

Return Value

<int>

The handle of the interpreter.

CSR_ERROR if the interpreter table is full.

Special Notes

The event interpreter's structure is added to the event manager's interpreter list and will be considered to be available at all times. The module in which the interpreter routine resides must remove the event interpreter if it terminates without remaining resident. The interpreter routine specified by the INTERP structure is assumed to have an assembly interface.

The added event interpreter routine must preserve the ES segment register.

The added event interpreter should not make direct calls into DOS.

See Also

INTERP structure listed above.

DeskMate Technical Reference Event Manager

event_delete_edriver (handle)

int handle;

event_delete_edriver removes an event driver from the event manager's driver list.

Input Parameters

handle is the handle of the event driver to delete as returned by **event_add_edriver**.

Return Value

None.

Special Notes

event_delete_edriver must be made prior to the module containing the event driver terminating and leaving memory.

See Also

event_add_edriver()

DeskMate Technical Reference

Event Manager

event_delete_interp (handle)

int handle;

event_delete_interp removes a CSR user interpreter.

Input Parameters

<int>

handle is the handle of the event interpreter to delete as returned by **event_add_interp**.

Return Value

None.

Special Notes

event_delete_interp must be made prior to the module containing the event interpreter terminating and leaving memory.

See Also

event_add_interp()

DeskMate Technical Reference

Event Manager

event_get_uninterp (pEVENT)
EVENT *pEVENT;

event_get_uninterp determines the uninterpreted event that generated the last event returned by the event manager.

Input Parameters

pEVENT is a pointer to an EVENT structure.

Return Value

None.

Special Notes

Technical Description:

The event manager process events with a two step procedure: 1) get the current base or raw event and 2) interpret that event. The result is an interpreted event. **event_get_uninterp** returns the uninterpreted or raw event which was used to derive the interpreted event. An uninterpreted event may come from one of three sources: 1) a built-in or added device driver (a raw event), 2) the message queue (a message event resulting from an **event_write**), or 3) the current raw event (a raw event from a device driver that has only been scanned.) Assuming that no "unusual" event drivers have been added, **event_get_uninterp** will return a raw event or a message event. An outside or command event cannot be returned by this service (outside events cannot be written to the message queue.) A command event may be returned, but only if the event came from the message queue. Raw events (mouse or character events) may come from any of the three uninterpreted event sources.

Please note the mouse events will have device coordinates and not world coordinates. If they are to be used with world coordinates and window relative comparisons they must be converted using the **vid_dcx_to_wcx**, and **vid_dcy_to_wcy** calls.

Non-technical Description:

If you get an outside or command event and you simply must know what key or mouse action caused that event, then use this service.

See Also

vid_dcx_to_wcx()
vid_dcy_to_wcy()
event_write

DeskMate Technical Reference

Event Manager

event_purge ()

event_purge removes the current event from the event queue.

Input Parameters

None.

Return Value

None.

Special Notes

An example of when **event_purge** is used is when an event has occurred outside the application's currently active window and the application chooses not to service the event.

If an event was written (via **event_write**) while a new event was in the event queue, only the written event will be purged.

event_purge will stop any mouse hold processing, until the mouse button is released.

Example

```
event_purge();
```

DeskMate Technical Reference

Event Manager

event_read (pEVENT)
EVENT *pEVENT;

event_read returns an event.

Input Parameters

pEVENT is a pointer to an EVENT structure.

Return Value

None.

Special Notes

event_read waits for an event from the user and fills the specified structure. The event is removed from the event queue when read, unless it is an outside event. If a main menu bar has been defined via the **mb_draw** routine, then **event_read** will process that menu bar for the application and any other interpreters.

A task switch event will be returned to child windows differently based on whether the user invokes the task switch via the F10 Accessory Menu or with the ALT+= accelerator. If the task switch is invoked via the F10 Accessory Menu only the base window (e.g. the window in which the main menu bar was created) will receive the task switch event. All other windows will receive an outside event. However, if ALT+= is used the task switch event will be returned to all windows. No window will receive an outside event.

Example

```
EVENT EventStruct;  
EVENT *pEvent;  
  
pEvent = &EventStruct;  
  
/* read an event */  
event_read( pEvent );
```

DeskMate Technical Reference Event Manager

event_scan (pEVENT)
EVENT *pEVENT;

event_scan polls for an event.

Input Parameters

pEVENT is a pointer to an EVENT structure.

Return Value

None.

Special Notes

event_scan fills the specified structure with the current event. The event is **NOT** removed from the event queue. If no event is available, then the specified EVENT structure will contain an EVENT_NULL for the message type upon return to the application. If a main menu bar has been defined via the **mb_draw** routine, then **event_scan** will process that menu bar for the application.

A task switch event will be returned to child windows differently based on whether the user invokes the task switch via the F10 Accessory Menu or with the ALT+= accelerator. If the task switch is invoked via the F10 Accessory Menu only the base window (e.g. the window in which the main menu bar was created) will receive the task switch event. All other windows will receive an outside event. However, if ALT+= is used the task switch event will be returned to all windows. No window will receive an outside event.

Example

```
EVENT EventStruct;  
EVENT *pEvent;  
  
pEvent = &EventStruct;  
/* scan for an event */  
event_scan( pEvent );
```

DeskMate Technical Reference

Event Manager

event_write (pEVENT)
EVENT *pEVENT;

event_write places the specified event on the event queue.

Input Parameters

pEVENT is a pointer to an **EVENT** structure.

Return Value

None.

Special Notes

event_write places an event into the event queue, usually for the purpose of "passing" the event to another window. The specified event message type and parameter will be returned to the next call to **event_read** or **event_scan** exactly as defined and regardless of the window that the event is read from. Mouse events will re-evaluate the x and y elements for the next caller, but will **NOT** evaluate outside events. If the specified event message type is **EVENT_REEVALUATE**, then the current "raw" event will be re-evaluated and returned to the next caller. This will cause a full evaluation for the next caller, including evaluation of outside events. In this case, the next caller will **NOT** receive an event message type and parameter as defined in the specified structure.

If the message type of the specified event is not an **EVENT_REEVALUATE**, any pending event in the event queue will not be affected by **event_write**. The event written will be returned to the next caller, e.g. a written event will have priority over a pending event. Once the written event has been removed from the event queues the pending event will be returned upon the next **event_read** or **event_scan**.

Example

```
EVENT EventStruct;  
EVENT *pEvent;  
  
pEvent = &EventStruct;  
/* write an event */  
event_write( pEvent );
```

File I/O Manager

"File I/O Manager"

Table of Contents

General Description/Notes	11- 1
Structures/Defines	11- 3
Current_Disk	Returns the current active disk.....11- 7
Delete_File	Deletes a file.....11- 8
dlgbox_Generic	Displays OPEN/MERGE/LOAD dialog box.....11- 9
dlgbox_Run	Displays the RUN dialog box.....11-10
dlgbox_Run_this	Display RUN DB, prompts filled..... 11-11
dlgbox_SaveAs	Displays the SAVE AS dialog box.....11-12
dosENDFIND.....	Restores old Disk Transfer Area to DTA..11-13
dosFINDFIRST.....	Attempts to locate a file name match....11-14
dosFINDNEXT.....	Locates next file or directory match....11-15
dosSTARTFIND.....	Establishes a new Disk Transfer Area....11-16
file_already_exists ...	Determines if a file exists.....11-17
FillWithDir	Fills a list box with a set of files....11-18
fil_access	Returns file attributes.....11-19
fil_close	Closes a file.....11-20
fil_close_error_msg ...	Displays close error messages.....11-21
fil_commit	Writes buffered file data to disk.....11-22
fil_create	Creates a file.....11-23
fil_create_error_msg ..	Displays create error messages.....11-24
fil_dup	Duplicates a file handle.....11-25
fil_force_close	Forces DOS to close a file.....11-26
fil_lock_bytes	Locks specified portion of file.....11-27
fil_lseek	Seeks to an offset in a file.....11-28
fil_lseek_error_msg ...	Displays lseek error messages.....11-29
fil_menu_merge	Merges a file.....11-30
fil_menu_new	Takes the application to a "new" state..11-32
fil_menu_open	Prompts the user for file to open.....11-34
fil_menu_page	Runs the page setup accessory.....11-36
fil_menu_quit	Sets up the datafile so app may exit...11-37
fil_menu_run	Prompts user to run of alternate app...11-38
fil_menu_save	Saves applications data in memory.....11-39
fil_menu_saveas	Prompts user for filename to save data..11-40

<code>fil_open</code>Opens a file.....	11-41
<code>fil_open_error_msg</code>Displays open error messages.....	11-43
<code>fil_read</code>Reads from a file.....	11-44
<code>fil_read_error_msg</code>Displays read error messages.....	11-45
<code>fil_read_far</code>Reads a file from far memory.....	11-46
<code>fil_unlock_bytes</code>Unlocks specified portion of a file.....	11-48
<code>fil_write</code>Writes to a file.....	11-49
<code>fil_write_error_msg</code>	...Displays write error messages.....	11-50
<code>fil_write_far</code>Writes a file from far memory.....	11-51
<code>Get_Directory</code>Gets the current directory.....	11-53
<code>get_drive_map</code>Gets bitmap of all drives in system.....	11-54
<code>get_free_space</code>Gets the amount of disk free space.....	11-55
<code>guf_bind_end</code>Terminates bindings to GUF resources....	11-56
<code>guf_bind_init</code>Initializes bindings to GUF resources...	11-57
<code>is_floppy</code>Determines if a drive is removable.....	11-58
<code>msgbox_SaveChanges</code>Displays the SAVE CHANGES message box...	11-59
<code>Path_Expand</code>Expands a partial path.....	11-60
<code>prguf_bind_end</code>Terminate bindings to low-level GUF.....	11-62
<code>prguf_bind_init</code>Initializes bindings to low-level GUF...	11-63
<code>qfn</code>Qualifies a path as existing in drive...	11-64
<code>Rename_File</code>Renames a file.....	11-66
<code>Select_Disk</code>Selects the specified drive.....	11-67
<code>Set_CPU_Speed</code>Sets the CPU speed.....	11-68
<code>Set_Directory</code>Sets the current directory as specified.	11-69
<code>valid_drive</code>Determines if specified drive is valid..	11-70
<code>valid_filename</code>Verifies correctness of filename chars..	11-71
<code>valid_filename_wild</code>	...Verifies filename using wildcard char...	11-72

DeskMate Technical Reference

File I/O Manager

General Description/Notes

DeskMate File I/O provides three levels of I/O support, including low level direct access, high level block oriented support including user interface management, and a database.

In order to use these File I/O routines the user must call either **guf_bind_init** or **prguf_bind_init** depending upon the level of support the application needs. See below for an explanation of which resources contain which functions.

Low Level File I/O

The low level file I/O consists of 8 calls which perform direct file access such as opening, reading, and writing disk files. These familiar calls allow the application complete freedom to manage data files in any way necessary. At the same time the application must fully perform tasks such as interacting with the user, error handling and disk swapping. The only errors handled by the low level calls are critical disk I/O errors such as the user leaving the drive door open, bad media errors or write protect errors.

If **dmerrno** contains a value of **DM_ERROR** (-1) it means that a critical error (other than write protect) occurred and the user chose the CANCEL action from the "Disk not ready" message box.

If **dmerrno** contains a value of -2, it means that the user chose the CANCEL action from the "Write protect" message box.

It is necessary for the application to extern the variable "**dmerrno**" if it is using the low level I/O calls, so that it may check if an error occurs during any of the I/O calls. The critical error routine will be set up by desk for all applications. Desk will also be responsible for displaying a RETRY/CANCEL message box in the event a critical I/O error occurs. If the application makes a low level call and is returned a critical error value, it should call **fil_force_close** to force the closure of the file and allow DOS to properly free its resources.

High Level File I/O

The high level I/O support consists of eight calls which perform the standard "File menu" functions. (New, Open..., Save, etc) for applications's whose data is in a single contiguous block. These calls completely manage the low level calls, including all interaction with the user and all error handling. They also maintain a consistent user interface for the "file menu" functions across all applications. High level file I/O is very useful because the application does not have to develop any file I/O functions at all with the exception of managing these calls.

High level I/O also performs an optional service of saving the current page setup with each individual datafile, and storing an identifier with each data file so that application's can distinguish data files in a way other than by the file extension. This file identifier and the page setup information is stored in a header which is automatically created at the beginning of each data file and the management of this header information is completely invisible to the application.

Miscellaneous File I/O Routines

These are some general file manipulation routines which are provided for use by the application.

Database File I/O

The database file I/O is discussed further in the database section of this manual.

DeskMate Technical Reference

File I/O Manager

1989 Summary of changes to the File I/O

In 1989 DeskMate 03.03.0x the GUF resource was split into two resources, the high level functions contained in DMGUF.RES, and the low level functions contained in PRGUF.RES. There are separate initialization calls for each different part. When binding to the high level resource (DMGUF.RES), the low level resource (PRGUF.RES) binding is automatic. The application only need bind to the resource it uses. Below is a list of the different functions contained in the different resources.

Functions included in the high level GUF, DMGUF.RES, which require **guf_bind_init**:

dlgbox_Generic	dlg_Run
dlgbox_SaveAs	dlg_Run_this(new for '89)
fil_close_error_msg	fil_create_error_msg
fil_lseek_error_msg	fil_menu_merge
fil_menu_new	fil_menu_open
fil_menu_page	fil_menu_quit
fil_menu_run	fil_menu_save
fil_menu_saveas	fil_open_error_msg
fil_read_error_msg	fil_write_error_msg
file_already_exists	FillWithDir
msgbox_SaveModifiedData	msgbox_SaveChanges
Set_CPU_Speed	valid_filename
valid_filename_wild(new for '89)	

Functions included in the low level GUF, PRGUF.RES, which require **prguf_bind_init**:

Current_Disk	Delete_File
dosENDFIND(new for '89)	dosFINDFIRST(new for '89)
dosFINDNEXT(new for '89)	dosSTARTFIND(new for '89)
fil_access	fil_close
fil_create	fil_dup
fil_force_close	fil_lock_bytes
fil_lseek	fil_open
fil_read	fil_read_far(new for '89)
fil_unlock_bytes	fil_write
fil_write_far(new for '89)	get_free_space
Get_Directory	get_drive_map
is_floppy	Path_Expand
qfn(new for '89)	Rename_File
Select_Disk	valid_drive

DeskMate Technical Reference

File I/O Manager

Structures/Defines

```

/*-----*/
/*
/* DMGUF.H contains the File I/O structures and defines */
/*
/*-----*/

WILD          1
NO_WILD       0

FILE_HEADER_LENGTH      22

/* file access modes */
OPEN_FOR_READ      0
OPEN_FOR_WRITE     1
OPEN_FOR_UPDATE    2

/* file sharing modes ( OR with a file access mode from above ) */
EXCLUSIVE          0x90
DENY_WRITE         0xA0
DENY_READ          0xB0
DENY_NONE          0xC0 /* share */

OPEN_FOR_UPDATE_SHARE 0xA2
OPEN_READ_DENY_NONE  0xC0

OPEN_NO_DIALOG      1      /* do not use a dialog box on open */
OPEN_WITH_DIALOG    0      /* use a dialog box on the open */

/* file menu merge */
MERGE_NO_DIALOG     1
MERGE_WITH_DIALOG   0

LOAD_WITH_DIALOG    2
LOAD_NO_DIALOG      3

/* used for dlgbox_Generic */
FIO_OPEN            1
FIO_MERGE           2
FIO_LOAD            3

FIO_CHAR_XEXT       100     /* these are the same in dmcsr.h */
FIO_CHAR_YEXT       220     /* same as in dmcsr.h world coordinate */
FIO_SCREEN_WIDTH    (80 * FIO_CHAR_XEXT)
FIO_SCREEN_HEIGHT   (25 * FIO_CHAR_YEXT)
FIO_PB_YEXT         (2 * FIO_CHAR_YEXT)

MAX_FILE_NAME_SIZE  65      /* maximum size that a filename can be */
MAX_FILES           50      /* used in the FillWithDir call */
MAX_BUFFER_SIZE_FOR_FILES 650 /* MAX_FILES * 13 */

/* MAPRECT of dlgbox_Generic */
op_col 26 * FIO_CHAR_XEXT + FIO_CHAR_XEXT/2      /* starting column */
op_row 7 * FIO_CHAR_YEXT                          /* starting row */
op_width 27 * FIO_CHAR_XEXT                      /* width */
op_height 12 * FIO_CHAR_YEXT                     /* height */

sa_col 24 * FIO_CHAR_XEXT /* starting column for dlgbox SaveAs */
sa_row 9 * FIO_CHAR_YEXT  /* starting row for dlgbox SaveAs */
sa_width 32 * FIO_CHAR_XEXT /* width of dlgbox SaveAs */
sa_height 7 * FIO_CHAR_YEXT /* height of dlgbox SaveAs */

```

DeskMate Technical Reference

File I/O Manager

```

FIO_RUN_WIDTH      (36 * FIO_CHAR_XEXT)
FIO_RUN_HEIGHT     ((6 * FIO_CHAR_XEXT) + FIO_PB_YEXT)
FIO_RUN_XORG        ((FIO_SCREEN_WIDTH - FIO_RUN_WIDTH) >> 1)
FIO_RUN_YORG        ((FIO_SCREEN_HEIGHT - FIO_RUN_HEIGHT) >> 1)

FIO_PRINT_WIDTH     (35 * FIO_CHAR_XEXT)
FIO_PRINT_HEIGHT    ((11 * FIO_CHAR_XEXT / 2) + FIO_PB_YEXT)
FIO_PRINT_XORG       ((FIO_SCREEN_WIDTH - FIO_PRINT_WIDTH) >> 1)
FIO_PRINT_YORG       ((FIO_SCREEN_HEIGHT - FIO_PRINT_HEIGHT) >> 1)

/* file i/o error codes */
DMERR_INVALID_START_POINT      1
DMERR_FILE_NOT_FOUND           2
DMERR_INVALID_FILENAME         3
DMERR_TOO_MANY_OPEN_FILES      4
DMERR_INVALID_PATHNAME         5
DMERR_INVALID_FILE_HANDLE      6
DMERR_INVALID_ACCESS_CODE      12
DMERR_OUT_OF_DISK_SPACE        28
DMERR_FILE_LOCKED              32
DMERR_READ_ONLY_FILE           65

DMERR_NONDESTRUCTIVE_ABORT      101
DMERR_DESTRUCTIVE_ABORT         102
DMERR_INVALID_FILE_TYPE         103
DMERR_INVALID_LOCK_LENGTH       104
DMERR_NO_EEPROM                 105
DMERR_INVALID_DRIVE             106
DMERR_INVALID_DIRECTORY         107
DMERR_BASE_TOO_LONG             108
DMERR_EXTENSION_TOO_LONG        109
DMERR_INVALID_FILENAME_CHAR     110
DMERR_NO_BASE                   111
DMERR_EXPANDED_PATH_TOO_LONG    112

DM_ERROR      -1

/* file types */
PERSONAL_TEXT_FILE      1
OLD_WORKSHEET_FILE      2
FILER_FILE              3
PERSONAL_TEXT_ASCII_FILE 10
MAIL_FILE               11
MUSIC_FILE              12
OFFICE_TEXT_FILE        13
WORKSHEET_FILE          14
OFFICE_TEXT_ASCII_FILE  15
DRAW_FILE               16
CALENDAR_FILE           17
TELECOM_FILE            18
PAINT_FILE              19
DRAW88_FILE             20
VENDOR_TAG              50

```

DeskMate Technical Reference

File I/O Manager

```
/* this is the datafile structure */
typedef struct datafile_defn
{
    unsigned char FileType;          /* what application owns the file */
    unsigned char Modified;          /* flag for save data prompt, */
                                    /* set to TRUE or FALSE */
    int    FileHandle;              /* file handle, -1 if no open */
                                    /* file (untitled) */
    long    FileSize;               /* number of bytes in file */
    char    *pFilename;             /* ptr to filename string */
    char    *pExtension;            /* ptr to application's filename ext */
    char    *pStart;                /* ptr to beginning of data block to write */
    char    *pEnd;                  /* ptr to end of data block to write */
    char    *pTop;                  /* ptr to top of available memory */
    char    *pTmpfil;               /* temporary filename pointer */
} DATAFILE;

typedef struct mergefile_defn
{
    unsigned char FileType;          /* what application owns the file */
    unsigned char Modified;          /* flag for save data prompt, */
                                    /* TRUE/FALSE */
    int    FileHandle;              /* file handle, -1 if no open file */
                                    /* (untitled) */
    long    FileSize;               /* number of bytes in file */
    char    *pFilename;             /* ptr to filename string */
    char    *pExtension;            /* ptr to application's filename */
                                    /* extension */
    char    *pStart;                /* ptr to beginning of data block to write */
    char    *pEnd;                  /* ptr to end of data block to write */
    char    *pTop;                  /* ptr to top of available memory */
    char    *pTmpfil;               /* temporary filename pointer */
} MERGEFILE;
```

The high level I/O calls require a pointer to a DATAFILE structure containing information about the current data file.

DATAFILE.FileType:

Identifies the application which this data file belongs to. The FileType is checked before the file is loaded to see that the file belongs to the application. If FileType is equal to zero, no checking is performed on the selected file.

DATAFILE.Modified:

This should be set to TRUE if the data file has been modified since the last time that the data file has been saved. A value of FALSE indicates that the data file has not been changed, and is also the state after calls to new, open, save, and saveas are made. When any keystroke or user action affects the data file, the application must set this to TRUE.

DATAFILE.FileHandle:

This is the handle to the data file that is set when the data file is opened, and is used to access the data file on subsequent high level I/O calls. If a call to open the data file is unsuccessful, or after closing a data file, this will be set to DM_ERROR. After performing a NEW, the application should set this value to DM_ERROR to indicate that no data file is open.

DATAFILE.FileSize:

This is the length (in bytes) of the data file. The application does not need to initialize or update this parameter.

DeskMate Technical Reference

File I/O Manager

DATAFILE.pFilename:

This is a pointer to a null terminated filename of the current working data file. The application should initially set this to point to a null string. It is not necessary for the application to maintain this pointer after initially setting it, except as noted in pTmpfil below.

DATAFILE.pExtension:

This is a pointer to a null terminated filename extension representing the extension used by the application's data file. This should be initialized by the application when it first loads and begins running. This is used for identifying data files that belong to the current application. This string must be 3 characters + null terminator.

DATAFILE.pStart:

Pointer to the beginning of the application's data block.

DATAFILE.pEnd:

Pointer to the current end of the application's data block. When the high level I/O writes the data block to the disk, it writes all the data starting at pStart and ending at pEnd.

DATAFILE.pTop:

Pointer to the last available byte of the application's data block area. The high level I/O uses this pointer to check if the data file is too large to be loaded by the application.

DATAFILE.pTmpfil: Before a data file is opened, the filename pointed to by pFilename is copied to this buffer (the buffer then holds the filename entered by the user in the open, load, or merge dialog box). This is provided so that an application may (if it wishes) force the high level I/O call to load the datafile even though the file type and extension do not match those normally used by the application. When the filetypes do not match, the high level I/O routines will return an error DM_ERROR and dmerrno will be set to DMERR_INVALID_FILE_TYPE. If the application wishes to force the datafile to be opened, it must set FileType in it's DATAFILE structure to the value of NULL (a flag specifying IGNORE the filetype) and copy the filename pointed to by pTmpfil back to the buffer pointed to by pFilename in it's DATAFILE structure. If the application then calls the I/O routine again, the datafile will be opened and loaded.

The following is an example of how an application might initialize the datafile structure upon loading and running:

```
char Filename[64] = ""; /* example null terminated filename */
char Tempname[64]; /* example Tmpfil file name buffer */
char Extension[] = "DOC"; /* example null terminated extension */
char Workspace[1024]; /* example 1K data buffer */
DATAFILE datafile;
datafile.FileType = TEXT_FILE;
datafile.Modified = FALSE;
datafile.FileHandle = -1;
datafile.pFilename = &Filename[0];
datafile.pExtension = &Extension[0];
datafile.pStart = &Workspace[0];
datafile.pTop = &Workspace[1023];
datafile.pTmpfil = &Tempname[0];
```

DeskMate Technical Reference

File I/O Manager

Current_Disk ()

Current_Disk returns the current active disk to the caller.

Input Parameters

None.

Return Value

<char>

The current active disk in the form of A, B, C, etc.

Example

```
char CurrentDisk;  
  
/* return the current active disk (e.g. 'A', 'B', 'C', etc) */  
CurrentDisk = Current_Disk();
```

DeskMate Technical Reference

File I/O Manager

Delete_File (pFilename)

char *pFilename;

Delete_File removes the directory entry for the specified filename, thus deleting the file.

Input Parameters

pFilename is a pointer to a character string containing the name of the file to be removed.

Return Value

<int>

NULL if successful.

DM_ERROR if an error occurs.

Special Notes

Read-only files **CANNOT** be deleted by this call. Network Access Rights: Requires Create access rights.

If DM_ERROR is returned, dmerrno will be set to one of the following:

- Two (2) if the file was not found
- Three (3) if the path was not found
- Five (5) if access was denied.

Example

```
int ReturnCode;  
  
/* delete the file "SAMPLE.TXT" */  
ReturnCode = Delete_File("SAMPLE.TXT");
```

DeskMate Technical Reference

File I/O Manager

dlgbox_Generic (pExtension, pFilename, bFlag)

char *pExtension;
char *pFilename;
int bFlag;

dlgbox_Generic provides the application's with a consistent way to prompt the user for a filename to OPEN, MERGE, or LOAD.

Input Parameters

pExtension is a pointer to the applications extension.

pFilename is a pointer to the buffer area to receive the filename.

bFlag is either FIO_OPEN, FIO_MERGE, or FIO_LOAD depending upon what the application wants the title of the box to read.

Return Value

TRUE and **pFilename** should point to a valid filename if successful.

FALSE if the user took the CANCEL action.

Special Notes

pFilename is usually defined by pDataFile->pFilename. The length of this field must be defined as MAX_FILE_NAME_SIZE because the full path is returned to the application.

If **pFilename** contains a full path the dialog box parses it and changes to the appropriate directory and displays the filename in the edit field. Upon exit the dialog box builds a full path (including drive) and puts it where **pFilename** points, then attempts to change the directory back to what it was when **dlgbox_Generic** was called.

pExtension is usually defined by pDataFile->pExtension. This is used as a wildcard mask when loading the files into the list box. It is also used as the default extension to append to the end of the filename if the user does not specify an extension. If a period is specified at the end of the filename no extension will be appended.

Example

```
char FileName[MAX_FILE_NAME_SIZE];  
int ReturnCode;  
  
/* put up a dialog box that allows the      */  
/* user to select a file from a list of      */  
/* files with the extension "DOC". The      */  
/* dialog box is titled for opening a        */  
/* file. The name of the file to open      */  
/* is returned in the buffer FileName.      */  
ReturnCode = dlgbox_Generic ( "DOC", &FileName[0], FIO_OPEN );
```

See Also

DATAFILE structure in dmguf.h

DeskMate Technical Reference

File I/O Manager

dlgbox_Run ()

dlgbox_Run provides applications a consistent way to prompt the user for the program name and data name they want to execute next.

Input Parameters

None.

Return Value

TRUE is returned if the user entered a valid program name and chose the OK action.

FALSE is returned if the user takes the CANCEL action.

Special Notes

The specified program name will be run instead of DESKTOP.PDM. Only programs with extensions of COM, EXE, BAT and PDM are valid.

The full path and drive specified are registered with the core through **dm_SetNextApp** so the program will be executed when the current program exits. The program name is converted to uppercase but the datafile is not, since the user may want to pass flags, options, or switches.

Example

```
int ReturnCode;

/* display a dialog box that will prompt the user */
/* for the program name and data file name. */
ReturnCode = dlgbox_Run();
```

See Also

dlgbox_Run_this()
dm_SetNextApp()

DeskMate Technical Reference

File I/O Manager

1989 DeskMate 03.03.0x ONLY

dlgbox_Run_this (pFilename, pDatafile)

char *pFilename;

char *pDatafile;

dlgbox_Run_this performs the same function as **dlgbox_Run**, prompting the user for the program name and data file name they wish to execute next. In addition, the strings pointed to by **pFilename** and **pDatafile** are already displayed in the dialog box edit fields.

Input Parameters

pFilename is a pointer to a character string containing the path name of the executable file.

pDatafile is a pointer to a character string containing the data filename associated with the program.

Return Value

TRUE is returned if the user entered a valid program name and chose the OK action.

FALSE is returned if the user takes the CANCEL action.

Special Notes

The specified program name will be run instead of DESKTOP.PDM. Only programs with extensions of COM, EXE, BAT and PDM are valid.

The full path and drive specified are registered with DeskMate through **dm_SetNextApp** so the program will be executed when the current program exits. The program name is converted to uppercase but the datafile is not, since the user may want to pass flags, options, or switches.

Example

```
int ReturnCode;
char *pFilename = {"TEXT.PDM"};
char *pDatafile = {"SAMPLE.DOC"};

/* display a dialog box with "TEXT.PDM" in the program edit field */
/* and "SAMPLE.DOC" in the data file edit field */
ReturnCode = dlgbox_Run_this( pFilename, pDatafile );
```

See Also

dlgbox_Run()
dm_SetNextApp()

DeskMate Technical Reference

File I/O Manager

dlgbox_SaveAs (pDatafile, pFilename)

DATAFILE *pDatafile;
char *pFilename;

dlgbox_SaveAs provides the applications with a consistent way to prompt the user for the name of the file to be saved.

Input Parameters

pDatafile is a pointer to a DATAFILE structure defining the data to be saved.

pFilename is a pointer to a character string to store the filename in, which must be at least MAX_FILE_NAME_SIZE bytes long.

Return Value

<int>

TRUE if the filename is successfully entered and accepted.

FALSE if the user selects CANCEL from the "Save As" dialog box or from the "File Already Exists, Overwrite?" dialog box.

Special Notes

If a file already exists by the same name the user is prompted to overwrite with Yes, No, or Cancel options.

If the file does not exist, a check is made to see if the file will fit on the disk. If it will fit, the file is created and then closed. Therefore, a file of zero length was created and closed with the name stored in **pFilename**, thus zeroing the FileSize element in the DATAFILE structure. The data is not written to the file at this time. **dlgbox_SaveAs** is generally used by applications which do their own file I/O via the low level calls. It is the responsibility of the application to close the file associated with **pDatafile** and open up the new file and write to it. The required items in the DATAFILE structure are: FileType, pEnd, pStart. pEnd and pStart are used to calculate the amount of space needed on the disk.

Example

```
DATAFILE    TEXTFILE;  
DATAFILE    *pTextFile;  
char   pTextFileName[MAX_FILE_NAME_SIZE];  
  
pTextFile = &TEXTFILE;  
dlgbox_SaveAs ( pTextFile, pTextFileName );
```

See Also

fil_menu_saveas()
DATAFILE structure in dmgu.f.h

DeskMate Technical Reference

File I/O Manager

1989 DeskMate 03.03.0x ONLY

dosENDFIND ()

dosENDFIND restores the old Disk Transfer Area to the DTA specified before a **dosSTARTFIND** was called.

Input Parameters

None.

Return Value

None.

Special Notes

dosENDFIND must be called after **dosSTARTFIND**, **dosFINDFIRST** and **dosFINDNEXT** calls are made. If a **dosSTARTFIND** call is made without a later **dosENDFIND** call, the DOS system DTA will be incorrect.

Example

```
/* restore the DTA after find first and find next calls */  
dosENDFIND();
```

See Also

dosSTARTFIND()
dosFINDFIRST()
dosFINDNEXT()

Only works when running under DM303 or later.

DeskMate Technical Reference

File I/O Manager

1989 DeskMate 03.03.0x ONLY

dosFINDFIRST (pPathname, Attribute)

```
char *pPathname;  
int Attribute;
```

dosFINDFIRST attempts to locate a file name match with the name pointed to by **pPathname**.

Input Parameters

pPathname is a pointer to either a full pathname or filename which will usually contain wildcards. **Attribute** is the file attribute for which a match is being sought. When set to zero, **dosFINDFIRST** locates normal files (files with no special attributes). If **Attribute** is set to 16 decimal, **dosFINDFIRST** will search for the first sub-directory match.

Return Value

DM_ERROR if no file (or directory, if specified) match was made or if a critical error was encountered during the operation.

If no error occurred, the first file or directory match is stored as a null-terminated string in the Disk Transfer Area (DTA) that was established by **dosSTARTFIND**.

Special Notes

Before calling **dosFINDFIRST**, the DTA must be set by calling **dosSTARTFIND**. Otherwise, results may be unpredictable.

In order for this function to be useful, the pathname specified will usually contain "*" or "?" wildcards.

Example #1

```
int ReturnCode;  
char *pPathname = {"*.DOC"};  
int Attribute = 0;  
  
/* Find first filename with "DOC" extension */  
ReturnCode = dosFINDFIRST( pPathname, Attribute );
```

*Only works when running
under DM303 or later.*

Example #2

```
int ReturnCode;  
char *pPathname = {"C:\\*"};  
int Attribute = 16;  
  
/* Find first sub-directory entry from the root */  
ReturnCode = dosFINDFIRST( pPathname, Attribute );
```

See Also

dosSTARTFIND()
dosENDFIND()
dosFINDNEXT()

DeskMate Technical Reference

File I/O Manager

1989 DeskMate 03.03.0x ONLY

dosFINDNEXT ()

dosFINDNEXT locates the next file or directory name match (if any) specified in the previous **dosFINDFIRST**.

Input Parameters

None.

Return Value

DM_ERROR if no file (or directory, if specified) match was made or if a critical error was encountered during the operation.

If no error occurred, the first file or directory match is stored as a null-terminated string in the Disk Transfer Area (DTA) that was established by **dosSTARTFIND**.

Special Notes

Before calling **dosFINDFIRST**, the DTA must be set by calling **dosSTARTFIND**. Otherwise, results may be unpredictable.

In order for this function to be useful, the pathname specified will usually contain "*" or "?" wildcards.

Example

```
int    ReturnCode;
```

```
/* Find the next filename match associated with dosFINDFIRST */  
/* Be sure dosSTARTFIND and dosFINDFIRST have already been called */  
ReturnCode = dosFINDNEXT();
```

See Also

dosSTARTFIND()
dosFINDFIRST()
dosENDFIND()

*Only works when running under
MM303 or later.*

DeskMate Technical Reference File I/O Manager

1989 DeskMate 03.03.0x ONLY

dosSTARTFIND (pDTA)

char *pDTA;

dosSTARTFIND establishes a new Disk Transfer Area, necessary for calling **dosFINDFIRST** (and subsequently **dosFINDNEXT**).

Input Parameters

pDTA is a pointer to the Disk Transfer Area to be used during **dosFINDFIRST** and **dosFINDNEXT**.

Return Value

None.

Special Notes

dosENDFIND must be called after a **dosSTARTFIND**, **dosFINDFIRST** and **dosFINDNEXT**. If **dosSTARTFIND** is called without a later **dosENDFIND** the DOS system DTA will be incorrect.

Example

```
char AppDTA[128];  
  
/* Establish the Disk Transfer Area for find first and find next */  
dosSTARTFIND( AppDTA );
```

See Also

dosENDFIND()
dosFINDFIRST()
dosFINDNEXT()

*Only works when running under
DM303 or later.*

DeskMate Technical Reference

File I/O Manager

file_already_exists (pFilename)
char *pFilename;

file_already_exists verifies whether a file exists.

Input Parameters

pFilename is a pointer to a character string which contains the filename.

Return Value

<int>

TRUE if the file exists.

FALSE if the file does not exist.

Example

```
char *pFilename;  
  
if( file_already_exists( pFilename ) == FALSE )  
{  
    /*the file does NOT exist */  
}
```


DeskMate Technical Reference

File I/O Manager

FillWithDir (pWildcard, ppIndex, pFiles, Max, Dirswi)

```
char *pWildcard;  
char **ppIndex;  
char *pFiles;  
int Max;  
int Dirswi;
```

FillWithDir provides the applications a way to fill a list box with a grouped set of files.

Input Parameters

pWildcard is a character pointer to the wildcard mask to be searched upon.

ppIndex is a pointer to a buffer area where the application wants the list of pointers to the files.

pFiles is a pointer to the buffer where the names of the files will actually reside.

Max is an integer of the number of files that you have room for, this is needed to calculate the size of you buffer areas.

Dirswi is a flag when zero (0) then only files that match the wildcard are displayed. If **Dirswi** is 10 hex then directories that match the wildcard are displayed.

Return Value

<int>

The number of files found will never be more than **Max**.

Special Notes

pWildcard should point to wildcards such as **"*.*"** to show all files on the disk, **"*.c"** to show all files that have a **"c"** extension, etc.

ppIndex is a pointer to a list of pointers that point into **pFiles**.

pFiles should provide enough room for all of the files that you expect to receive.

Max should be set to **MAX_FILES** upon entry.

Dirswi when using this option **pWildcard** should point to **"*.*"** to get all directories. Directories are displayed in brackets, **"[..]"** for example. The current directory **"."** is not displayed, but the parent directory and any sub directories are displayed.

Example

```
char *FileIndex[MAX_FILES];  
char FileNames[MAX_BUFFER_SIZE_FOR_FILES];  
int NumFilesFound;  
  
/* fill the buffer FileNames with all files      */  
/* that have the extension "c" and return        */  
/* the number of files that match. FileIndex    */  
/* points to the beginning of each name in the  */  
/* FileNames buffer.                            */  
NumFilesFound = FillWithDir ( "*.c", &FileIndex[0],  
                             &FileNames[0], MAX_FILES, 0 );
```

DeskMate Technical Reference File I/O Manager

Bug: Won't work for root directories.

```
fil_access ( pdmerrno, pDataFileName ) } WRONG. Should be:  
int *pdmerrno; } fil_access ( pDataFileName );  
char *pDataFileName;
```

fil_access returns the file attributes of the file with the specified name.

Input Parameters

pdmerrno is the address of the dmerrno variable in which file I/O errors are returned.

pDataFileName is a pointer to a null terminated string specifying the name of the file in which to get the attributes. This string may be a complete or incomplete pathlist.

Return Value

<int>

The lower byte of the integer returned contains the attributes of the file specified if the file exists on the disk in the specified path and no error occurred.

DM_ERROR if an error occurred or if the file does not exist in the specified path.

Special Notes

A special use of **fil_access** may be to determine if the user has swapped disks while an application is active and expects the data disk to remain in the drive. An important example of this are applications that use the DeskMate database. They should check to make sure the data file is still present in the drive before making database calls because the database assumes that this is true without checking itself, and if the disk is swapped, then the database will write over the top of whatever disk is in the drive, which can potentially destroy the contents of the swapped disk.

If DM_ERROR is returned, dmerrno may have any one of the following values:

DMERR_FILE_NOT_FOUND if the file was not found.

DMERR_INVALID_FILENAME if the pathname was invalid.

a value of -1 or -2 if a critical error occurred.

The following is how to interpret the attribute byte:

bit 0 set if the file is read only

bit 1 set if the file is a hidden file

bit 2 set if the file is a system file

bit 3 set if the file is a volume label

bit 4 set if the file is a sub-directory

bit 5 set if the file has been written to and closed

Example

```
char FileAttribute;
```

```
/* get the file attributes for "SAMPLE.TXT" file. */  
FileAttribute = fil_access("SAMPLE.TXT");
```

DeskMate Technical Reference

File I/O Manager

fil_close (FileHandle)
int FileHandle;

fil_close closes the specified disk file.

Input Parameters

FileHandle is the handle of the desired file that was assigned to the file when the file was opened with **fil_open**.

Return Value

<int>

DM_ERROR if an error occurred closing the disk file.

Special Notes

If DM_ERROR is returned, and the value of dmerrno is a -1 or -2 (critical error), then the application must call **fil_force_close** to properly close the file and free up the DOS file resources.

If DM_ERROR is returned, dmerrno may have any one of the following values:

DMERR_INVALID_FILE_HANDLE if the specified file handle has not been assigned.
a value of -1 or -2 if a critical error occurred.

Example

```
int ReturnCode;  
int FileHandle;  
  
/* close the file whose handle is FileHandle */  
ReturnCode = fil_close(FileHandle);
```

See Also

fil_open()
fil_force_close()

DeskMate Technical Reference

File I/O Manager

fil_close_error_msg ()

fil_close_error_msg displays the file close error message box after an error in **fil_close**.

Input Parameters

None.

Return Value

None.

Special Notes

fil_close_error_msg uses the current value of **dmerrno** to determine which error message to display. **fil_close_error_msg** must be used immediately after the error occurred (before any other file I/O call is made) to insure that the error message reflects the actual condition. If a critical error occurred, no message will be displayed because the critical error handler displayed a message.

Example

```
/* display the error that just occurred */  
/* when trying to close the file */  
fil_close_error_msg();
```

DeskMate Technical Reference

File I/O Manager

fil_commit (pdmerrno, FileHandle)

int *pdmerrno;

int FileHandle;

fil_commit causes all buffered data, for a file, to be written to the device.

Input Parameters

pdmerrno is the address of the dmerrno variable in which file I/O errors are returned.

FileHandle is a file handle which was returned from a **fil_open**, or **fil_create** call.

Return Value

None.

Special Notes

If the DOS version is 3.0 or greater, function 068h will be used. For DOS version less than 3.0 the calls to **dup_fil** (045h) and **close_file** (03eh) will be called. If **dmerrno** is equal to **DM_ERROR** upon entry, **fil_commit** does nothing since **DM_ERROR** indicates a critical error has occurred.

Example

```
int dmerrno;
```

```
int FileHandle;
```

```
fil_commit ( &dmerrno, FileHandle );
```

DeskMate Technical Reference

File I/O Manager

fil_create (pDataFileName)
char *pDataFileName;

fil_create creates a disk file of the specified name.

Input Parameters

pDataFileName is a pointer to a string specifying the name of the file to create. This string may be a complete or incomplete pathlist.

Return Value

<int>

The handle of the created file if no error occurred.

DM_ERROR if an error occurred creating the specified file.

Special Notes

If the file is created without an error, then the file remains open for READ/WRITE access. The READ/WRITE pointer is set to the first byte of the file.

If DM_ERROR is returned, dmerrno may have any one of the following values:

DMERR_READ_ONLY_FILE if the file is a read only file.

DMERR_FILE_LOCKED if the file is locked.

DMERR_INVALID_FILENAME if the filename was invalid.

DMERR_TOO_MANY_OPEN_FILES if the number of files currently open is the same as the number of files specified by the "files=" command in CONFIG.SYS.

DMERR_INVALID_PATHNAME if the pathname was invalid.

DMERR_DIRECTORY_FULL if the directory is full.

a value of -1 or -2 if a critical error occurred.

Example

```
int FileHandle;
```

```
/* create the file "SAMPLE.TXT" and save the file handle */  
FileHandle = fil_create("SAMPLE.TXT");
```

DeskMate Technical Reference

File I/O Manager

fil_create_error_msg ()

fil_create_error_msg displays the file create error message box after an error in **fil_create**.

Input Parameters

None.

Return Value

None.

Special Notes

fil_create_error_msg uses the current value of **dmerrno** to determine which error message to display. **fil_create_error_msg** must be used immediately after the error occurred (before any other file I/O call is made) to insure that the error message reflects the actual condition.

If a critical error occurred, no message will be displayed because the critical error handler displayed a message.

Example

```
/* display the error that just occurred */  
/* when trying to create the file */  
fil_create_error_msg();
```

DeskMate Technical Reference

File I/O Manager

fil_dup (FileHandle)
int FileHandle;

fil_dup takes an already opened file handle and returns a new handle that refers to the same file at the same position.

Input Parameters

FileHandle is the handle of the file you wish to duplicate. The one which was assigned to the file when the file was opened with **fil_open**.

Return Value

<int>

An integer representing a new handle if successful.
DM_ERROR if unsuccessful.

Special Notes

If DM_ERROR is returned, dmerrno will be set to one of the following values:

DMERR_INVALID_FILE_HANDLE if the specified file handle has not been assigned.
DMERR_TOO_MANY_OPEN_FILES if the number of files currently open is the same as the number of files specified by the "files=" command in CONFIG.SYS.

Example

```
int FileHandle;  
int DupFileHandle;  
  
/* Duplicate a handle whose filehandle is FileHandle */  
DupFileHandle = fil_dup ( FileHandle );
```

See Also

fil_open()

DeskMate Technical Reference

File I/O Manager

fil_force_close (FileHandle)
int FileHandle;

fil_force_close forces DOS to close the specified file, even though DOS cannot access the file.

Input Parameters

FileHandle is the handle of the desired file that was assigned to the file when the file was opened with **fil_open**.

Return Value

None.

Special Notes

fil_force_close should be called if a **fil_close** returns on a critical error, or if a **fil_read** or **fil_write** returns on a critical error code and the application knows it must immediately close the file.

fil_force_close may be called without first calling **fil_close**, but **MUST NOT** be called unless a critical error has occurred.

Example

```
int FileHandle;  
  
/* a critical error has occurred, force the file */  
/* with the handle of FileHandle to close */  
fil_force_close(FileHandle);
```

See Also

fil_open()
fil_close()
fil_read()
fil_write()

DeskMate Technical Reference

File I/O Manager

```
int fil_lock_bytes ( FileHandle, Offset, Length )
int FileHandle;
long Offset, Length;
```

fil_lock_bytes allows a user to keep other users from accessing a portion of a file for a temporary period.

Input Parameters

FileHandle is a valid open file handle.

Offset is the offset into file to begin locking.

Length is the number of bytes to be locked.

Return Value

NULL if successful.

DM_ERROR if an error occurred.

Special Notes

If DM_ERROR is returned, dmerrno is set to normal DOS values or to a value of DMERR_INVALID_LOCK_LENGTH if **Length** is 0.

fil_lock_bytes will make the DOS function locking call.

The lock and unlock functions should only be used when a file is opened using the DENY_READ or DENY_NONE sharing modes. This is available with DOS versions 3.0 or greater.

Example

```
int FileHandle; /* open file handle */
long Offset = 100; /* file offset to begin locking */
long Length = 500; /* number of bytes to lock */

erc = fil_lock_bytes( FileHandle, Offset, Length );
if( erc == 0 ) /* 500 characters beginning at 100 are now locked */
;
else /* dmerrno is set */
```

See Also

fil_unlock_bytes()
fil_open()

DeskMate Technical Reference

File I/O Manager

fil_lseek (FileHandle, SeekOffset, StartOffset)

int FileHandle;
long SeekOffset;
int StartOffset;

fil_lseek causes the disk drive to seek to the specified offset in the specified disk file.

Input Parameters

FileHandle is the handle of the desired file that was assigned to the file when the file was opened with **fil_open**.

SeekOffset is the number of bytes from the **StartOffset** to seek to in the file. **SeekOffset** can be negative or positive (although negative is not valid if **StartOffset** is equal to zero). If **SeekOffset** is a negative number the seek is performed to that many bytes before the **StartOffset**.

StartOffset is a code specifying the offset from the start of the file that the **SeekOffset** is calculated from. (See Special Notes section below)

Return Value

<long>

The number of bytes from the beginning of the file to the new file pointer location.

-1L if an error occurred seeking to the specified offset in the file.

Special Notes

The **StartOffset** may be one of the following:

Zero (0) starts seeking from the beginning of file.

One (1) starts seeking from the current file pointer location.

Two (2) starts seeking from the end of the file.

If **DM_ERROR** is returned, **dmerrno** may have any one of the following values:

DMERR_INVALID_START_POINT if the **StartOffset** code is not one of the previously specified values.

DMERR_INVALID_FILE_HANDLE if the specified file handle has not been assigned.

a value of -1 or -2 if a critical error occurred.

Example

```
long fil_lseek( int FileHandle, long SeekOffset, int StartOffset );
int FileHandle;
long NumBytes;

/* move the file pointer to an offset of 0 */
/* bytes from the beginning of the file with */
/* the handle FileHandle. */
NumBytes = fil_lseek(FileHandle, 0L, 0);
```

See Also

fil_open()

DeskMate Technical Reference

File I/O Manager

fil_lseek_error_msg ()

fil_lseek_error_msg displays the file lseek error message box after an error in **fil_lseek**.

Input Parameters

None.

Return Value

None.

Special Notes

fil_lseek_error_msg uses the current value of **dmerrno** to determine which error message to display. **fil_lseek_error_msg** must be used immediately after the error occurred (before any other file I/O call is made) to insure that the error message reflects the actual condition. If a critical error occurred, no message will be displayed because the critical error handler displayed a message.

Example

```
/* display the error that just occurred */  
/* when trying to lseek into the file */  
fil_lseek_error_msg();
```

DeskMate Technical Reference

File I/O Manager

fil_menu_merge (pDatafile, bDialogFlag)

DATAFILE *pDatafile;

int bDialogFlag;

fil_menu_merge displays the "file merge" (or "load file") dialog box allowing the user to specify the name of the file to merge (or load), and reads the data into memory.

Input Parameters

pDatafile is a pointer to a DATAFILE structure which provides information about the area of memory available for merging/loading a file and the application's file type and extension.

bDialogFlag may have a value of **MERGE_WITH_DIALOG**, causing the "merge file" dialog box to appear, or a value of **LOAD_WITH_DIALOG**, causing the "load file" dialog box to appear, both allowing the user to choose the filename to load. A value of **MERGE_NO_DIALOG** or **LOAD_NO_DIALOG** cause the load to occur without prompting the user for the name of the file to load, and instead use the file name specified in the DATAFILE structure.

Return Value

<unsigned int>

The number of bytes read into the data buffer if the data file was loaded successfully and the filesize is greater than zero.

DM_ERROR if an error occurred loading the data file from disk.

Special Notes

Whether **fil_menu_merge** is used to "Merge" or to "Load" a data file, the function performed is identical. The only difference is the title of the dialog box which appears to the user. **fil_menu_merge** prompts the user for the file name (if specified by **bDialogFlag**), opens the file, loads the file into memory at the address specified by **DATAFILE.pStart** and closes the file. The application is responsible for setting the pointers (**DATAFILE.pStart** and **DATAFILE.pTop**) which identify the area of memory reserved for merging/loading a file. In the case of merging, the application must manage data movement and pointer assignment to provide for proper insertion of the merged file and prevent existing data from being overwritten by merging in a file.

fil_menu_open prompts the user to save changes, **file_menu_merge** does not. **fil_menu_open** also leaves the file open after loading it, and **fil_menu_merge** closes the file after loading.

If a value of **DM_ERROR** is returned, the application should check **dmerrno** to see if the abort affected the data currently in memory. If **dmerrno** is a value of **DMERR_NONDESTRUCTIVE_ABORT**, then the data in memory was not destroyed by the read and the application may continue processing the data currently in memory. If **dmerrno** is a value of **DMERR_DESTRUCTIVE_ABORT**, then the data in the application's buffer may have been destroyed by the read and the application should set itself to the NEW "untitled" state.

DeskMate Technical Reference

File I/O Manager

Example

```
char DataBuffer[1000];
DATAFILE DataFileStruct;
long NumBytesRead;

/* merge data from a file into the last half of the buffer */
DataFileStruct.pStart = &DataBuffer[500];
DataFileStruct.pTop = &DataBuffer[999];

/* select a file to merge through a dialog box, read */
/* the file and fill the buffer with the data read. */
NumBytesRead = fil_menu_merge ( &DataFileStruct, MERGE_WITH_DIALOG );
```

See Also

DATAFILE structure in dmguif.h
fil_menu_open()

DeskMate Technical Reference

File I/O Manager

fil_menu_new (pDatafile)
DATAFILE *pDatafile;

fil_menu_new saves the application's data if it has been modified since the last time it was saved. (after prompting: "Save changes?" YES/NO/CANCEL)

Input Parameters

pDatafile is a pointer to the DATAFILE structure for the application's current data file.

Return Value

<int>

TRUE if the data was saved successfully or did not need to be saved.

DM_ERROR is returned if an error occurred.

Special Notes

If the Modified flag (in DATAFILE) is set to TRUE, the user will be prompted with "Save changes?" YES/NO/CANCEL.

If the data is saved successfully, the file is closed, the Modified flag is set to FALSE, and the datafile filename is set to NULL.

If TRUE is returned, the application has responsibility for performing the tasks to return to the NEW default state.

If DM_ERROR is returned the datafile could not be saved. The filename will have been set to NULL if the file could not be found to save to changes to it, and the file has been closed.

Example

```
char FileName[64] = "";
char FileExtension[4] = "DOC";
char TempName[64];
char TextBuffer[500];
int ReturnCode;
```

```
DATAFILE DataFileStruct =
{
    PERSONAL_TEXT_FILE, /* what application owns the file */
    FALSE,               /* flag for save data prompt, TRUE/FALSE */
    -1,                  /* file handle, -1 if no open file */
    /* (untitled) */
    0,                   /* number of bytes in file */
    &FileName[0],         /* ptr to filename string */
    &FileExtension[0],    /* ptr to application's filename ext */
    &TextBuffer[0],       /* ptr to beginning of data block to */
    /* write */
    &TextBuffer[499],    /* ptr to end of data block to write */
    &TextBuffer[499],    /* ptr to top of available memory */
    &TempName[0]         /* temporary filename pointer */
};
```

DeskMate Technical Reference

File I/O Manager

```
/* create a new data file for the text application. */  
/* since there is not a file currently opened, the user will */  
/* not be prompted to save the file. */  
ReturnCode = fil_menu_new ( &DataFileStruct );
```


DeskMate Technical Reference

File I/O Manager

fil_menu_open (pDatafile, bDialogFlag)
DATAFILE *pDatafile;
int bDialogFlag;

fil_menu_open saves the application data currently in memory to the disk if it has been changed since the last save (after prompting: "Save changes?" YES/NO/CANCEL) prompts the user for the name of the data file to open, if required, and opens the specified file and loads it into memory.

Input Parameters

pDatafile is a pointer to the DATAFILE structure which provides information about the application's data file currently in memory and the amount of memory available for a new file.

bDialogFlag may have a value of OPEN_WITH_DIALOG causing the open file dialog box to appear allowing the user to choose the filename to open, or may have a value of OPEN_NO_DIALOG.

Return Value

<unsigned int>

The number of bytes read into the data buffer if the open and read was successful and the filesize is greater than zero.

DM_ERROR if an error occurred during the open or read.

Special Notes

If the data file currently in memory when the function is called is successfully saved to the disk before loading the new file, the Modified flag will be set to FALSE.

If a value of DM_ERROR is returned, the application should check dmerrno to see if the error affected the data file currently in memory. If dmerrno is a value of DMERR_NONDESTRUCTIVE_ABORT, then the data in memory was not destroyed by the read and the application may continue processing the data currently in memory. If dmerrno is a value of DMERR_DESTRUCTIVE_ABORT, then the data in the application's buffer may have been destroyed by the read, and the application should set itself to a NEW "untitled" state.

If the data file is loaded successfully, the Modified flag is set to FALSE, the buffer pointed to by pDatafile->pFilename now contains the filename, and the file handle is set to the new file handle. The data file will only be loaded into memory if the data file length is greater than zero.

If the new data file cannot be opened, locked, or read, the Modified flag will not be changed, the previously opened file will be reopened, and DM_ERROR will be returned. If the previously opened file cannot be reopened, then the filename will be reset to "untitled" (a null string).

bDialogFlag controls whether or not the "open file" dialog appears. In most instances the dialog box is necessary to communicate with the user. However if the application is invoked from the DeskMate DeskTop, or the Run dialog box, and is passed a file name, then the "open file" dialog should not appear when the application calls **fil_menu_open**, to load the data file. It is in these special instances when the application knows the name of the file to be opened before making the call, **bDialogFlag** should be set to OPEN_NO_DIALOG.

If an error occurs and dmerrno is set to DMERR_INVALID_FILE_TYPE, it means the application's data file extension or FileType parameter does not match.

DeskMate Technical Reference

File I/O Manager

Example

```
char FileName[64] = "CURRENT";
char FileExtension[4] = "TXT";
char TempName[64];
char TextBuffer[500];
int ReturnCode;
long NumBytes;
int FileHandle;

DATAFILE DataFileStruct =
{
    PERSONAL_TEXT_FILE,      /* what application owns the file */
    TRUE,                    /* flag for save data prompt, TRUE/FALSE */
    0,                       /* file handle, -1 if no open file (untitled) */
    0L,                     /* number of bytes in file */
    &FileName[0],            /* ptr to filename string */
    &FileExtension[0],       /* ptr to application's filename */
                             /* extension */
    &TextBuffer[0],         /* ptr to beginning of data block */
                             /* to write */
    &TextBuffer[499],       /* ptr to end of data block to write */
    &TextBuffer[499],       /* ptr to top of available memory */
    &TempName[0],           /* temporary filename pointer */
};

/* set the file handle to the handle of the currently opened file */
DataFileStruct.FileHandle = FileHandle;

/* set the number of bytes in the file currently opened */
DataFileStruct.FileSize = NumBytes;

/* save the buffer to the current text file, prompt user for which */
/* file to open through a dialog box , and read the file into a */
/* buffer */
ReturnCode = file_menu_open ( &DataFileStruct, OPEN_WITH_DIALOG );
```

See Also

DATAFILE structure in dmguf.h
fil_menu_merge()
fil_menu_run()

DeskMate Technical Reference

File I/O Manager

fil_menu_page ()

fil_menu_page runs the page setup accessory.

Input Parameters

None.

Return Value

Zero if the accessory failed to load.

RETURN_KEY (000Dh) if the OK push button was selected from the accessory.

ESC_KEY (001B) if the CANCEL push button was selected from the accessory.

Special Notes

fil_menu_page should be called when the user chooses "Page setup..." from the application's file menu.

fil_menu_page does not save the screen area where the accessory will be displayed, and therefore it is the responsibility of the application to redraw the screen after calling **fil_menu_page**.

Example

```
/* run the page setup accessory */  
fil_menu_page();
```

DeskMate Technical Reference

File I/O Manager

fil_menu_quit (pDatafile)
DATAFILE *pDatafile;

fil_menu_quit saves the application's data currently in memory if it has been modified since the last time that it was saved (after prompting: "Save changes?" YES/NO/CANCEL), so that the application may exit.

Input Parameters

pDatafile is a pointer to a DATAFILE structure which provides information about the application's data file currently in memory.

Return Value

<int>

TRUE if the applications data was saved successfully or did not need to be saved (Modified == FALSE).

DM_ERROR if an error occurred saving the application's data to the disk.

Special Notes

If the application's data was saved to the disk successfully, the DATAFILE.Modified flag will be reset to a value of FALSE, and it is the application's responsibility to perform the tasks required to exit the application.

Example

```
DATAFILE DataFileStruct;  
int ReturnCode;  
  
/* prompt user to see if they want to */  
/* save the changes made to the file */  
DataFileStruct.Modified = TRUE;  
/* save the buffer in memory to the */  
/* current data file and close the file */  
ReturnCode = fil_menu_quit ( &DataFileStruct );
```

See Also

DATAFILE structure in dmguif.h
fil_menu_run()

DeskMate Technical Reference

File I/O Manager

fil_menu_run (pDataFile)
DATAFILE *pDataFile;

fil_menu_run saves the application's data currently in memory if it has been modified since the last save (after prompting: "Save changes?" YES/NO/CANCEL), and then displays the run dialog box and allowing the user to specify a program and data file to run after exiting the application rather than returning to the DeskMate DeskTop.

Input Parameters

pDataFile is a pointer to a DATAFILE structure which provides information about the application's data file currently in memory.

Return Value

TRUE if the application's data was saved successfully (or did not need to be saved), and the user entered a valid program name and chose OK.

DM_ERROR if an error occurred saving the application's data to the disk, or if the user chose the CANCEL option from from the "Save Changes" message box or from the "Run" dialog box.

Special Notes

fil_menu_run functions the same way as **fil_menu_quit**, except the "Run" dialog box appears for the user to specify the next application to run. The specified application and data file are registered automatically with the desk executive through **dm_SetNextApp**.

Example

```
DATAFILE DataFileStruct;  
int ReturnCode;  
  
/* prompt user to see if they want to */  
/* save the changes made to the file */  
DataFileStruct.Modified = TRUE;  
/* save the buffer in memory to the current file */  
/* and close the file. then run another application */  
/* specified by the user. */  
ReturnCode = fil_menu_run ( &DataFileStruct );
```

See Also

DATAFILE structure in dmguif.h
fil_menu_quit()
dm_SetNextApp()

DeskMate Technical Reference

File I/O Manager

fil_menu_save (pDatafile)
DATAFILE *pDatafile;

fil_menu_save saves the application's current data file in memory.

Input Parameters

pDatafile is a pointer to the DATAFILE structure which provides information about the application's data file currently in memory.

Return Value

<int>

TRUE if the applications data was saved successfully.

DM_ERROR if an error occurred saving the application's data to the disk.

Special Notes

If the file exists on the disk and the data in memory is now smaller than the file on the disk, the file is closed and reopened with truncation to recapture disk space. Regardless of the file size, the file is written out to the disk, then closed to flush the read/write buffer, and then reopened. As long as the process is not interrupted by an I/O error or by another user getting access to the file, the filename will remain the same. Otherwise it will be set to "untitled" (a null string). If the lock on the file is lost in this process, then the filename is changed to NULL.

If an error occurs or there is not enough disk space to save the data, dmerrno is set to DMERR_NONDESTRUCTIVE_ABORT. If the file is not found on the disk the user is prompted to swap disks, create a new file on the current disk, or cancel the save function. If cancel is chosen the filename is set to a null string and dmerrno is set to DMERR_NONDESTRUCTIVE_ABORT.

Example

```
DATAFILE DataFileStruct;  
int ReturnCode;  
  
/* save the buffer in memory to the current data file */  
ReturnCode = fil_menu_save ( &DataFileStruct );
```

See Also

DATAFILE structure in dmguif.h

DeskMate Technical Reference

File I/O Manager

fil_menu_saveas (pDatafile)
DATAFILE *pDatafile;

fil_menu_saveas displays the "save as" dialog box prompting the user for the name of the new file to save the data to and saves the application's data currently in memory to the disk under that new file name.

Input Parameters

pDatafile is a pointer to the DATAFILE structure which provides information about the applications data file currently in memory.

Return Value

<int>

TRUE if the applications data was saved successfully.

DM_ERROR if an error occurred saving the application's data to disk.

Special Notes

If an error occurs or there is not enough disk space on the disk to save the data, then dmerrno is set to DMERR_NONDESTRUCTIVE_ABORT.

If the save was successful the new filename is copied to the buffer pointed to by the DATAFILE.pFilename pointer.

If the file already exists, the user will be prompted whether or not to over write the file.

Example

```
DATAFILE DataFileStruct;  
int ReturnCode;  
  
/* save the buffer in memory to a new filename specified by the user */  
ReturnCode = fil_menu_saveas ( &DataFileStruct );
```

See Also

dlgbox_SaveAs()

DATAFILE structure in dmguif.h

DeskMate Technical Reference

File I/O Manager

fil_open (pDataFileName, Mode)
char *pDataFileName;
int Mode;

fil_open opens the specified file with the specified access attributes.

Input Parameters

pDataFileName is a pointer a string specifying the name of the file to open. This string may be a complete or incomplete pathlist.

Mode is an integer specifying the access and sharing attributes for the file.

Return Value

<int>

The file handle of the file which was opened if no error occurred.

DM_ERROR if an error occurred opening the specified file.

Special Notes

The **Mode** specifies the access and sharing attributes for the file to be opened. the **mode** value is determined by ORing together an access code and a sharing code. These codes are defined in dmguif.h.

The access code may be one of the following values:

OPEN_FOR_READ if the file is to have read-only access.

OPEN_FOR_WRITE if the file is to have write-only access.

OPEN_FOR_UPDATE if the file is to have both read and write access.

The sharing codes may be one of the following values:

EXCLUSIVE to specify other processes cannot open a file for READ or WRITE.

DENY_WRITE to specify other processes can open the file for READ only.

DENY_READ to specify other processes can open the file for WRITE only.

DENY_NONE to specify other processes can open the file for READ, WRITE or both.

If DM_ERROR is returned, dmerrno may have any one of the following values:

DMERR_READ_ONLY_FILE if the file is a read only file.

DMERR_FILE_LOCKED if the file is locked.

DMERR_FILE_NOT_FOUND if the file was not found.

DMERR_TOO_MANY_OPEN_FILES if the number of files currently open is the same as the number of files specified by the "files=" command in CONFIG.SYS.

DMERR_INVALID_PATHNAME if the pathname was invalid.

DMERR_INVALID_ACCESS_CODE if the specified file access attribute code is not one of the previously specified values.

a value of -1 or -2 if a critical error occurred.

DeskMate Technical Reference

File I/O Manager

Example

```
int FileHandle;  
  
/* open the file "SAMPLE.TXT" to read data and save the file handle */  
/* lock out all other processes from accessing this file */  
FileHandle = fil_open("SAMPLE.TXT", OPEN_FOR_READ | EXCLUSIVE);
```

See Also

fil_open_error_msg()

DeskMate Technical Reference

File I/O Manager

fil_open_error_msg ()

fil_open_error_msg displays the file open error message box after an error in **fil_open**.

Input Parameters

None.

Return Value

None.

Special Notes

fil_open_error_msg uses the current value of **dmerrno** to determine which error message to display. **fil_open_error_msg** must be used immediately after the error occurred (before any other file I/O call is made) to insure that the error message reflects the actual condition.

If a critical error occurred, no message will be displayed because the critical error handler displayed a message.

Example

```
/* display the error that just occurred */  
/* when trying to open the file */  
fil_open_error_msg();
```

DeskMate Technical Reference

File I/O Manager

fil_read (FileHandle, pDataBuffer, Length)

int FileHandle;

char *pDataBuffer;

unsigned int Length;

fil_read reads the specified number of bytes from the specified disk file into the specified data buffer.

Input Parameters

FileHandle is the handle of the desired file that was assigned to the file when the file was opened with **fil_open**.

pDataBuffer is a pointer to the buffer to read the file data into, must be at least **Length** bytes long.

Length specifies the maximum number of bytes to read from the file.

Return Value

<unsigned>

The number of bytes that were actually read.

DM_ERROR if an error occurred reading from the disk file.

Special Notes

If the number of bytes to read is greater than the length of the disk file, then the file will be read to the END OF FILE and no error will be returned. **fil_read** will return the actual length of the file.

If DM_ERROR is returned, dmerrno may have any one of the following values:

DMERR_INVALID_PATHNAME if the specified path cannot be found.

DMERR_INVALID_FILE_HANDLE if the specified file handle has not been assigned.

a value of -1 or -2 if a critical error occurred.

Due to the small memory model, the maximum number of bytes which can be read at one time is 65,534.

Example

```
int FileHandle;
char FileBuffer[100];
int ReturnCode;
```

```
/* fill FileBuffer by reading 100 bytes */
/* from the file with the handle FileHandle */
ReturnCode = fil_read(FileHandle, &FileBuffer[0], 100);
```

See Also

fil_open()

DeskMate Technical Reference

File I/O Manager

fil_read_error_msg ()

fil_read_error_msg displays the file read error message box after an error in **fil_read**.

Input Parameters

None.

Return Value

None.

Special Notes

fil_read_error_msg uses the current value of **dmermo** to determine which error message to display. **fil_read_error_msg** must be used immediately after the error occurred (before any other file I/O call is made) to insure that the error message reflects the actual condition.

If a critical error occurred, no message will be displayed because the critical error handler displayed a message.

Example

```
/* display the error that just occurred */  
/* when trying to read from the file */  
fil_read_error_msg();
```

DeskMate Technical Reference

File I/O Manager

1989 DeskMate 03.03.0x ONLY

fil_read_far (FileHandle, pDataBuffer, Length, ReadSegment)

int FileHandle;

char *pDataBuffer;

unsigned int Length;

unsigned int ReadSegment;

fil_read_far reads the specified number of bytes from the specified disk file into the specified data buffer pointed to by **pDataBuffer**, within the segment **ReadSegment**.

Input Parameters

FileHandle is the handle of the desired file that was assigned to the file when the file was opened with **fil_open**.

pDataBuffer is a pointer to the buffer to read the file data into, must be at least **Length** bytes long.

Length specifies the maximum number of bytes to read from the file.

ReadSegment is the segment portion of the far address specified by **ReadSegment** and offset by **pDataBuffer**.

Return Value

<unsigned>

The number of bytes that were actually read.

DM_ERROR if an error occurred reading from the disk file.

Special Notes

In order to call **fil_read_far**, a far pointer must point to the segment and offset of the read buffer area. This is accomplished by performing a **fmalloc**, allocating memory referenced by a far pointer. Using 'C' library calls **FP_SEG** and **FP_OFF**, the values for **ReadSegment** and **pDataBuffer** can be obtained (see example).

If the number of bytes to read is greater than the length of the disk file, then the file will be read to the END OF FILE and no error will be returned. **fil_read_far** will return the actual length of the file.

If **DM_ERROR** is returned, **dmerrno** may have any one of the following values:

DMERR_INVALID_PATHNAME if the specified path cannot be found.

DMERR_INVALID_FILE_HANDLE if the specified file handle has not been assigned.

DM_ERROR or DMERR_WRITE_PROTECT if a critical error occurred.

Due to the small memory model, the maximum number of bytes which can be read at one time is 65,534.

DeskMate Technical Reference

File I/O Manager

Example

```
/* read 45000 bytes into another segment */
int   FileHandle;
char  *pDataBuffer;
unsigned int Length;
unsigned int ReadSegment;
char far  *lpBuffer;
int   ReturnCode;

/* allocate 45000 byte buffer */
lpBuffer = fmalloc(45000);
/* initialize the segment */
ReadSegment = FP_SEG( lpBuffer );
/* initialize the offset */
pDataBuffer = FP_OFF( lpBuffer );
Length = 45000;
ReturnCode = fil_read_far( FileHandle,pDataBuffer,Length,ReadSegment );
```

See Also

fil_read()
fil_write_far()

DeskMate Technical Reference

File I/O Manager

```
int fil_unlock_bytes ( FileHandle, Offset, Length )  
int FileHandle;  
long Offset, Length;
```

fil_unlock_bytes allows a user to let other users access a portion of a file that was previously locked using **fil_lock_bytes**.

Input Parameters

FileHandle is a valid open file handle.

Offset is the offset into file to begin unlocking.

Length is the number of bytes to be unlocked.

Return Value

NULL if successful.

DM_ERROR for errors.

Special Notes

If DM_ERROR is returned, dmerrno is set to normal DOS error values or to a value of DMERR_INVALID_LOCK_LENGTH if **Length** is zero.

fil_unlock_bytes will make the DOS function unlocking call.

The lock and unlock functions should only be used when a file is opened using the DENY_READ or DENY_NONE sharing modes. This is available with DOS versions 3.0 or greater.

Example

```
int handle; /* open file handle */  
long offset = 100; /* file offset to begin unlocking */  
long Length = 500; /* number of bytes to unlock */  
  
erc = fil_unlock_bytes( FileHandle, Offset, Length );  
if ( erc == 0 ) /* 500 characters beginning at 100 are now unlocked */  
    ;  
else /* dmerrno is set */
```

See Also

fil_lock_bytes()
fil_open()

DeskMate Technical Reference

File I/O Manager

fil_write (FileHandle, pDataBuffer, Length)

int FileHandle;

char *pDataBuffer;

unsigned int Length;

fil_write writes the specified number of bytes from the specified data buffer to the specified disk file.

Input Parameters

FileHandle is the handle of the desired file that was assigned to the file when the file was opened with **fil_open**.

pDataBuffer is a pointer to the buffer to write the file data from.

Length specifies the number of bytes to write to the file.

Return Value

<unsigned>

The number of bytes actually written to the file if no disk error occurred.

DM_ERROR if an error occurred writing to the disk file.

Special Notes

If DM_ERROR is returned, dmermo may have any one of the following values:

DMERR_INVALID_PATHNAME if the specified path cannot be found.

DMERR_INVALID_FILE_HANDLE if the specified file handle has not been assigned.

DMERR_OUT_OF_DISK_SPACE if the disk is full.

DMERR_READ_ONLY_FILE if the file access is read only.

a value of -1 or -2 if a critical error occurred.

Due to the small memory model, the maximum number of bytes which can be written at one time is 65,534.

Example

```
int FileHandle;
```

```
char FileBuffer[100];
```

```
int ReturnCode;
```

```
/* write 100 bytes from FileBuffer to */
```

```
/* the file with the handle FileHandle */
```

```
ReturnCode = fil_write(FileHandle, &FileBuffer[0], 100);
```

See Also

fil_open()

DeskMate Technical Reference

File I/O Manager

fil_write_error_msg ()

fil_write_error_msg displays the file write error message box after an error in **fil_write**.

Input Parameters

None.

Return Value

None.

Special Notes

fil_write_error_msg uses the current value of **dmerrno** to determine which error message to display. **fil_write_error_msg** must be used immediately after the error occurred (before any other file I/O call is made) to insure that the error message reflects the actual condition.

If a critical error occurred, no message will be displayed because the critical error handler displayed a message.

Example

```
/* display the error that just occurred */  
/* when trying to write to the file */  
fil_write_error_msg();
```

DeskMate Technical Reference

File I/O Manager

1989 DeskMate 03.03.0x ONLY

fil_write_far (FileHandle, pDataBuffer, Length, WriteSegment)

int FileHandle;

char *pDataBuffer;

unsigned int Length;

unsigned int WriteSegment;

fil_write_far writes the specified number of bytes from the specified data buffer to the specified disk file within the segment specified by **WriteSegment**.

fil_write_far allows for a data buffer which is outside the default data segment of the application.

Input Parameters

FileHandle is the handle of the desired file that was assigned to the file when the file was opened with **fil_open**.

pDataBuffer is a pointer to the buffer containing the data to be written.

Length specifies the number of bytes to write to the file.

WriteSegment is the segment portion of the far address specified by **WriteSegment** and offset by **pDataBuffer**.

Return Value

<unsigned>

The number of bytes that were actually written to the file if no disk error occurred.

DM_ERROR if an error occurred writing to the disk file.

Special Notes

In order to call **fil_write_far**, a far pointer must point to the segment and offset of the write buffer area. This is accomplished by performing an **fmalloc**, allocating memory referenced by a far pointer. Using 'C' library calls **FP_SEG** and **FP_OFF**, the values for **WriteSegment** and **pDataBuffer** can be obtained (see Example).

If **DM_ERROR** is returned, **dmerrno** may have any one of the following values:

DMERR_INVALID_PATHNAME if the specified path cannot be found.

DMERR_INVALID_FILE_HANDLE if the specified file handle has not been assigned.

DMERR_OUT_OF_DISK_SPACE if the disk is full.

DM_ERROR or DMERR_WRITE_PROTECT if a critical error occurred.

Due to the small memory model, the maximum number of bytes which can be written at one time is 65,534.

DeskMate Technical Reference

File I/O Manager

Example

```
/* write 45000 bytes from another segment */
int   FileHandle;
char  *pDataBuffer;
unsigned int Length;
unsigned int ReadSegment;
char far  *lpBuffer;
int   ReturnCode;

/* allocate 45000 byte buffer */
lpBuffer = fmalloc(45000);
/* initialize the segment */
ReadSegment = FP_SEG( lpBuffer );
/* initialize the offset */
pDataBuffer = FP_OFF( lpBuffer );
Length = 45000;
ReturnCode = fil_write_far( FileHandle,pDataBuffer,Length,ReadSegment );
```

See Also

fil_write()
fil_read_far()

DeskMate Technical Reference

File I/O Manager

Get_Directory (Drive, pPath)

char Drive;
char *pPath;

Get_Directory returns the current directory for the specified drive.

Input Parameters

Drive is a character and must be in the form A, B, C, etc.

pPath must be defined as a pointer to a character string at least `MAX_FILE_NAME_SIZE` bytes long.

Return Value

Zero (0) means the current directory will be returned in the **pPath** parameter. The directory name will begin with a slash.

`DM_ERROR` if an invalid drive is specified, with `dmerrno` set to 15.

Example

```
char CurrentPath[MAX_FILE_NAME_SIZE]

/* get the current pathlist for drive C */
Get_Directory ( 'C', &CurrentPath[0] );
```

DeskMate Technical Reference

File I/O Manager

get_drive_map ()

get_drive_map returns a bitmap which identifies all valid drives in the system.

Input Parameters

None

Return Value

<unsigned long>

a bit map of the 26 drives, where A: = bit 0 (LSB), Z: = bit 25. If the bit is set it means a physical or redirected drive is mapped. If the bit is cleared then the drive is not mapped. Bits for "Virtual" drives are cleared.

Special Notes

ON DOS 3.0 OR HIGHER:

get_drive_map calls DOS function IOCTL, subfunction 0x0E (Get Logical Drive Map) for each of the 26 possible drives, to determine whether a physical or redirected drive exists. DOS "virtual" drives are disallowed.

ON DOS 2.x OR LOWER:

get_drive_map uses DOS Select Disk function to determine highest drive assigned. The equipment flag is checked specifically for B: only, so that we disallow the DOS virtual drive on single drive systems.

Example

```
unsigned long get_drive_map();
unsigned long ValidDrives;

/* get a bitmap of the current system's valid drives. */
/* the drives are returned as drive A = bit 0, drive Z = bit 25. */
ValidDrives = get_drive_map();
```

See Also

DOS technical reference manual.

DOS 2.x or Lower Select Disk call.

DOS 3.0 or Higher IOCTL call.

is_floppy()

DeskMate Technical Reference

File I/O Manager

get_free_space (Drive)
int Drive;

get_free_space returns a long number indicating the number of bytes free on the disk.

Input Parameters

Drive is an integer in the format 1=A, 2=B, etc.

Return Value

A long integer indicating the number of free bytes on the disk.

Example

```
long get_free_space();  
long FreeBytes;  
  
/* get the amount of free space on drive A */  
FreeBytes = get_free_space ( 1 );
```

This function can fail on large hard drives, returning a negative number.

DeskMate Technical Reference

File I/O Manager

guf_bind_end ()

In DeskMate 03.03.0x **guf_bind_end** terminates the applications bindings to both the low-level, (PRGUF.RES) and high-level, (DMGUF.RES) GUF resources.

In DeskMate 03.02.0x **guf_bind_end** terminates the applications bindings to the GUF resource.

Input Parameters

None.

Return Value

None.

Special Notes

guf_bind_end should be made when termination to the GUF resource(s) is desired by the application. Usually just prior to **csr_end**.

If the application makes this call it should *not* make the call to **prguf_bind_end**.

DeskMate Technical Reference

File I/O Manager

guf_bind_init ()

In 1989 DeskMate 03.03.0x **guf_bind_init** sets up the applications bindings to both the low-level, (PRGUF.RES) and high-level, (DMGUF.RES) GUF resources.

In 1988 DeskMate 03.02.00 **guf_bind_init** sets up the applications bindings to the the GUF resource.

Input Parameters

None.

Return Value

DM_ERROR if the high-level GUF resource, DMGUF.RES could not be loaded.

Special Notes

guf_bind_init should be called when the user wishes to use any of the calls in either of the GUF resources, usually right after **csr_init**.

If the application makes this call it should *not* make the call to **prguf_bind_init**.

See the General Description/Notes for a list of fileio manager calls contained in the GUF resource.

DeskMate Technical Reference

File I/O Manager

is_floppy (Drive)
char Drive;

is_floppy determines if the specified drive is removable.

Input Parameters

Drive is a character and must be in the form A,B,C etc.

Return Value

DM_ERROR if the drive is invalid.

TRUE if the drive is removable.

Zero (0) if the drive is fixed.

Example

```
if ( is_floppy (Drive) )  
    /* do floppy processing */
```

See Also

get_drive_map()

DeskMate Technical Reference

File I/O Manager

msgbox_SaveChanges ()

msgbox_SaveChanges displays a standard message box which states "Save Changes", YES/NO/CANCEL, and returns the users selection.

Input Parameters

None.

Return Value

MSG_YES if the user selected YES.
MSG_NO if the user selected NO.
MSG_CANCEL if the user selected CANCEL.

Example

```
if ( msgbox_SaveChanges() == MSG_YES )  
    /* save the data */
```

DeskMate Technical Reference

File I/O Manager

Path_Expand (pSource, pExpPath)

```
char *pSource;  
char *pExpPath;
```

Path_Expand takes a partial path, which may include "..", and expands it to a full pathname including drive specification. The filename portion of the path need not exist. **Path_Expand** also verifies that the filename portion does not contain too many characters in the base or extension, the drive specified is valid, and the expanded pathname is not longer than 64 characters. If a filename alone is specified it is appended to the end of the current expanded path.

Input Parameters

pSource is a pointer to a character string containing the filename or pathname to be expanded. **pExpPath** is a pointer to a buffer at least **MAX_FILE_NAME_SIZE**, which will contain the fully-expanded pathname.

Return Value

<int>

If the path was expanded without an error, the expanded pathname is returned at **pExpPath** and **NULL** is returned.

If **DM_ERROR** is returned, **dmerrno** may have any one of the following values:

- DMERR_INVALID_DRIVE** if the specified drive is not valid.
- DMERR_EXTENSION_TOO_LONG** if a sub-directory or filename extension is longer than 3 characters.
- DMERR_NO_BASE** if no basename was specified upon entry.
- DMERR_BASE_TOO_LONG** if a sub-directory or filename base is longer than 8 characters.
- DMERR_INVALID_PATHNAME** if **pSource** points to a path which cannot be expanded as specified or contains non-existing sub-directories.
- DMERR_EXPANDED_PATH_TOO_LONG** if the expanded pathname would become longer than 64 characters.
- DM_ERROR** or **DMERR_WRITE_PROTECT** if a critical error occurred.

Special Notes

The buffer pointed to by **pExpPath** should be at least **MAX_FILE_NAME_SIZE** bytes long. The partial path may include "..".

Path_Expand will not display any error messages to the user. All error messages are the responsibility of the application.

Path_Expand attempts to change and set directories while executing. If a floppy is not properly inserted in the drive specified or some other critical error occurs in **Path_Expand**, no message is given to the user and **dmerrno** will contain the appropriate error.

← = -2, not defined

DeskMate Technical Reference

File I/O Manager

Example

```
char *pSource = "..\\SAMPLE.DOC";
char pExpPath[MAX_FILE_NAME_SIZE];
int  ReturnCode;

/* expand from the current director of C:\\PATH1\\PATH2 */
/* and qualify the path */
ReturnCode = Path Expand( pSource, &ExpPath[0] );
/* upon return ExpPath will contain "C:\\PATH1\\PATH2\\SAMPLE.DOC" */
```

See Also

qfn()

DeskMate Technical Reference

File I/O Manager

1989 DeskMate 03.03.01 ONLY

prguf_bind_end ()

prguf_bind_end terminates the applications bindings to the low-level GUF resource PRGUF.RES.

Input Parameters

None.

Return Value

None.

Special Notes

prguf_bind_end should be made when termination to the low-level GUF resource, PRGUF.RES is desired by the application. Usually just prior to **csr_end**.

prguf_bind_end() is automatically made by **guf_bind_end**.

DeskMate Technical Reference

File I/O Manager

1989 DeskMate 03.03.01 ONLY

prguf_bind_init ()

prguf_bind_init sets up the bindings from the application to the low-level GUF resource, PRGUF.RES.

Input Parameters

None.

Return Value

DM_ERROR if the low-level GUF resource, PRGUF.RES could not be loaded.

Special Notes

prguf_bind_init should be called when the user wishes to use any of the calls in the low-level GUF resource, PRGUF.RES, usually right after **csr_init**.

prguf_bind_init() is automatically made by **guf_bind_init**.

DeskMate Technical Reference

File I/O Manager

1989 DeskMate 03.03.0x ONLY

qfn (pSource, pExpPath)
char *pSource;
char *pExpPath;

qfn qualifies a path as existing in the specified drive and sub-directories and fully expands the path. The filename portion of the path need not exist. **qfn** also verifies that the filename portion does not contain too many characters in the base or extension, the drive specified is valid, and the expanded pathname is not longer than 64 characters.

If a filename alone is specified it is appended to the end of the current expanded path.

Input Parameters

pSource is a pointer to a character string containing the filename or pathname to be expanded. **pExpPath** is a pointer to a buffer at least MAX_FILE_NAME_SIZE, which will contain the fully-expanded pathname.

Return Value

<int>

If the path was expanded without an error, the expanded pathname is returned at **pExpPath** and NULL is returned.

If DM_ERROR is returned, dmerrno may have any one of the following values:

DMERR_INVALID_DRIVE if the specified drive is not valid.

DMERR_EXTENSION_TOO_LONG if a sub-directory or filename extension is longer than 3 characters.

DMERR_BASE_TOO_LONG if a sub-directory or filename base is longer than 8 characters.

DMERR_INVALID_PATHNAME if **pSource** points to a path which cannot be expanded as specified or contains non-existing sub-directories.

DMERR_EXPANDED_PATH_TOO_LONG if the expanded pathname would become longer than 64 characters.

DM_ERROR or DMERR_WRITE_PROTECT if a critical error occurred.

Special Notes

The buffer pointed to by **pExpPath** should be at least MAX_FILE_NAME_SIZE bytes long. The partial path may include ".".

qfn will not display any error messages to the user. All error messages are the responsibility of the application.

qfn attempts to change and set directories while executing. If a floppy is not properly inserted in the drive specified or some other critical error occurs in **qfn**, no message is given to the user and dmerrno will contain the appropriate error.

DeskMate Technical Reference

File I/O Manager

Example

```
char *pSource = "..\\SAMPLE.DOC";
char pExpPath[MAX_FILE_NAME_SIZE];
int ReturnCode;

/* expand from the current director of C:\\PATH1\\PATH2 */
/* and qualify the path */
ReturnCode = qfn( pSource, &ExpPath[0] );
/* upon return ExpPath will contain "C:\\PATH1\\PATH2\\SAMPLE.DOC" */
```

See Also

Path_Expand()

*qfn() will lock up the system under DeskMate 3.02;
use Path_Expand().*

DeskMate Technical Reference

File I/O Manager

Rename_File (pOldname, pNewname)

char *pOldname;
char *pNewname;

Rename_File renames a file from **pOldname** to **pNewname**. Both files must reside on the same drive.

Input Parameters

pOldname is a pointer to a character string containing the old file name.
pNewname is a pointer to a character string containing the new file name.

Return Value

NULL if successful.
DM_ERROR if an error occurs.

Special Notes

Either or both pathnames may contain a drive specified in the form A:, B:, etc, and may contain a directory path, but both drives **MUST** be the same.

If DM_ERROR is returned, dmerrno will be set to one of the following:

- Three (3) if the path was not found,
- Five (5) if access was denied,
- Seventeen (17) if the files are not on the same device.

Example

```
int ReturnCode;  
  
/* rename the file "SAMPLE.TXT" to "NEW.TXT" */  
ReturnCode = Rename_File("SAMPLE.TXT", "NEW.TXT");
```

DeskMate Technical Reference

File I/O Manager

Select_Disk (Drive)

char Drive;

Select_Disk makes the specified drive the current drive.

Input Parameters

Drive is a character in the form of A, B, C, etc.

Return Value

None.

Special Notes

Because there is no way to test for errors, it is useful to call **valid_drive** or **get_drive_map** to verify the drive exists before calling **Select_Disk**.

Example

```
char NewDrive;  
Select_Disk ( NewDrive );
```

See Also

get_drive_map()
valid_drive()

DeskMate Technical Reference

File I/O Manager

Set_CPU_Speed (Speed)

int Speed;

Set_CPU_Speed allows for setting of the CPU speed.

Input Parameters

Speed is an integer with the value of "F" for fast and any other value for slow.

Return Value

None.

Special Notes

The default speed setting for machines with two speed CPU's is fast. It is suggested that slow only be used when an application requires a slower CPU speed.

Example

```
int     FAST = 'F';

/* set CPU speed to FAST */
Set_CPU_Speed ( FAST );
```

DeskMate Technical Reference

File I/O Manager

Set_Directory (pPathname)
char pPathname;

Set_Directory sets the current directory to the specified pathname.

Input Parameters

pPathname is a pointer to a character string containing the new current directory. It may contain a leading drive specification in the form A:, B:, etc.

Return Value

<int>
NULL if successful.
DM_ERROR if an error occurs.

Special Notes

If an error occurs dmerrno will be set to three (3) indicating the pathname is invalid.

Example

```
int    ReturnCode;  
  
/* change the current directory to c:\path1\path2 */  
ReturnCode = Set_Directory ( "C:\\PATH1\\PATH2" );
```

*This function may succeed, yet return an error.
The pathname should be converted to uppercase.
Specifying just a drive "C:" will succeed, but
return DM_ERROR.*

*The pathname should be expanded with
Path_Expand() before using this function.*

DeskMate Technical Reference

File I/O Manager

valid_drive (pDriveBuf)
char *pDriveBuf;

valid_drive takes a pointer to a drive specification (which may be followed by a pathname) and determines if the drive is valid.

Input Parameters

pDriveBuf is a pointer to a character string which contains the drive letter or a complete path.

Return Value

<int>

TRUE if the specified drive exists.

FALSE if the drive does not exist.

Special Notes

If the drive does not exist a message box is displayed stating "Invalid Drive" before returning FALSE.

The string pointed to by **pDriveBuf** is converted to uppercase.

Example

```
char *pDrive;  
  
if ( valid_drive ( pDrive ) == TRUE )  
{  
    /* drive does exist in the system */  
}
```

DeskMate Technical Reference

File I/O Manager

valid_filename (pFilename, pExtension)

char *pFilename;

char *pExtension;

valid_filename verifies all characters, and the form of a file name are correct. It appends the extension pointed to by **pExtension** to the file name, unless an extension is not desired, specified by file names ending with a period.

Input Parameters

pFilename is a pointer to a character string that contains the filename to be verified.

pExtension is a pointer to a character string that contains the default extension to be appended.

Return Value

<int>

TRUE if the filename is valid.

FALSE if the filename is invalid.

Special Notes

If the filename is invalid, before returning FALSE, a message box will be displayed stating the error condition:

- Invalid drive.

- Invalid Filename (Contains invalid characters).

- Filename base more than eight characters.

- Extension more than three characters.

pFilename may point to a full path including drive and directories.

The strings pointed to by **pFilename** and **pExtension** are converted to upper case.

If **pFilename** does not have an extension, and **pExtension** points to an empty string, **valid_filename** will still append a dot onto the string pointed to by **pFilename**.

Example

```
DATAFILE    TEXTFILE;  
DATAFILE    *pTextFile;
```

```
pTextFile = &TEXTFILE;
```

```
if ( valid_filename ( (pTextFile->pFilename, pTextFile->pExtension)  
                      == TRUE )  
{  
    /* the filename is valid */  
}
```

See Also

DATAFILE structure in dmguif.h

DeskMate Technical Reference

File I/O Manager

1989 DeskMate 03.03.0x ONLY

valid_filename_wild (pFilename, pExtension)

char *pFilename;

char *pExtension;

valid_filename_wild performs the same function as **valid_filename** with the exception that the wildcard characters "*" and "?" are accepted as valid filename characters.

Input Parameters

pFilename is a pointer to a character string that contains the filename to be verified.

pExtension is a pointer to a character string that contains the default extension to be appended.

Return Value

<int>

TRUE if the filename is valid.

FALSE if the filename is invalid.

Special Notes

If the filename is invalid, before returning FALSE, a message box will be displayed stating the error condition:

- Invalid drive.

- Invalid Filename (Contains invalid characters).

- Filename base more than eight characters.

- Extension more than three characters.

pFilename may point to a full path including drive and directories.

The strings pointed to by **pFilename** and **pExtension** are converted to upper case.

If **pFilename** does not have an extension, and **pExtension** points to an empty string, **valid_filename** will still append a dot onto the string pointed to by **pFilename**.

Example

```
DATAFILE    TEXTFILE;  
DATAFILE    *pTextFile;
```

```
pTextFile = &TEXTFILE;
```

```
if ( valid_filename_wild ( (pTextFile->pFilename, pTextFile->pExtension)  
                          == TRUE )  
{  
    /* the filename is valid */  
}
```

See Also

DATAFILE structure in dmguif.h

Form Resources

"Form Manager"

Table of Contents

Near Form Resource General Description/Notes.....	12- 1
Structures/Defines	12- 4
form_add_element ..Adds the specified element to the form.....	12-13
form_add_stroke ...Adds a stroke definition to the form.....	12-14
form_break_object .Reduces one object to constituent objects...	12-15
form_clear	Clears the form's display and stroke lists..12-16
form_compress	Compress free space from the form.....12-17
form_copy_element .Copies element to the clipboard.....	12-18
form_copy_group ...Copies group of elements to clipboard.....	12-19
form_cut_element ..Cuts the form element to the clipboard.....	12-20
form_cut_groupCuts a group of elements to clipboard.....	12-21
form_define_group ...Defines a group of elements for the form..	12-22
form_delete_element .Deletes element definition from form.....	12-23
form_delete_group ...Deletes all group elements from form.....	12-24
form_delete_stroke ..Deletes stroke character from form.....	12-25
form_delete_unused_strokes ...Deletes non-required strokes.....	12-26
form_display_region .Displays form elements in a region.....	12-27
form_dup_elementCreates a scaled copy of an element.....	12-28
form_dup_group	Creates a scaled copy of a group.....12-29
form_element_to_bottom	Moves element to bottom of defs....12-30
form_element_to_top	Moves element to top of defs.....12-31
form_expand	Expands the form display list.....12-32
form_find_element ...Finds an element in a rectangle.....	12-33
form_find_strokeFinds a stroke character.....	12-34
form_get_pointerGets a pointer to element definition.....	12-35
form_get_xy_extents .Gets maximum x,y values for all elements..	12-36
form_group_to_bottom	Moves group to bottom of defs.....12-37
form_group_to_top	Moves group to top of defs.....12-38
form_make_objectMakes an object from elements in group....	12-39
form_mod_attr	Modifies element attributes.....12-40
form_mod_grp_attr ...Modifies element attributes in the group..	12-41
form_move_element ...Moves and resizes an element.....	12-42
form_move_group	Moves and resizes a group of elements.....12-43

form_openInitializes the form buffer.....12-44
form_pasteAdds the clipboard display to the form....12-45
form_region_empty ...Determines if the form contains elements..12-46
form_replace_elementReplaces an element definition.....12-47
form_scroll_downScrolls the form down.....12-48
form_scroll_leftScrolls the form to the left.....12-49
form_scroll_right ...Scrolls the form to the right.....12-50
form_scroll_upScrolls the form up.....12-51
form_specify_group...Makes a list of element tags into a group.12-52
form_updateDraws/redraws the current form.....12-53

General Description/Notes

The forms resource calls are used to create pictures which are to be printed, manipulated, or stored for later use. Each "form" is a list of the graphic and text primitives which make up a picture. A form is made up of four contiguous parts, a header, a list of tags for each element entry, the element definitions themselves, and a list of definitions for any stroke font characters which are used in the form.

Each form header contains information used by the forms resource to maintain the form. Included are the size of the buffer allotted for the tag list and element definitions, and the size of the buffer allotted for the stroke definitions. Also included are pointers and counters used for the manipulation of the form, and a flag to allow or suppress output to the video display whenever elements are added or manipulated.

The tag list, or entry list, contains identifying tags for each of the elements in the form, along with the offset from the beginning of the buffer to the associated element's definition. The order of the entries in the entry list determines the order in which each element will be rendered to the display device. The entry list begins at the top of the buffer specified in the form header, and grows downward in memory. The element definitions begin at the bottom of the buffer specified in the form header, and grow upward towards the entry list.

Each element definition contains all information necessary to render the element onto a display device. All element's possess a bounding rectangle which specifies the upper left and lower right corner of the region in which the element will be displayed. All form coordinates are expressed in terms of world coordinates, allowing a form to encompass an area of 32K by 32K points. Element types include lines, rectangles, ellipses, polylines, and text strings. Text strings can be in either the system text font, or in application defined stroke fonts. In the case of stroke font text strings, the ASCII representation of the string will be stored in the element definition. The actual definition for the stroke characters will be stored in the form's stroke list.

The stroke list holds the definitions for all of the stroke font characters used in the form. This avoids storing duplicate definitions for characters in the element definition portion of the form. Each stroke character is defined as a series of lines (or strokes) drawn on a 128 by 128 point grid. The definition for the character is mapped into the bounding rectangle of the corresponding text element definition to allow the display of stroke text at any size or rotation (in 90 degree increments).

Enhanced Form Manager

The Enhanced Form Manager contains all of the functionality of the original form manager as well as font support. It will have its own set of bindings called **eform_bind_init** and **eform_bind_end**. The near form calls are still supported, using the **csr_bind_init** call, however they do not support fonts. The eform calls expect all pointer parameters to be far.

The **eform_bind_init** call contains a parameter that allows the font support to be turned off. If the flag is set to NULL the form manager will not be able to support fonts. If the flag is set to 1, then fonts are supported and applications will also have the ability to access the font engine or the enhanced video driver directly.

DeskMate Display List Format

The display list contains all of the information necessary to generate the form. The display list consists of two parts, the entries and the element definitions.

DeskMate Technical Reference Forms Resource

The form entries are simply an array of two-word entries. The first form entry contains the number of renderable elements in the form. All subsequent entries are an offset from the start of the form to the element definitions, and a tag which identifies the entry.

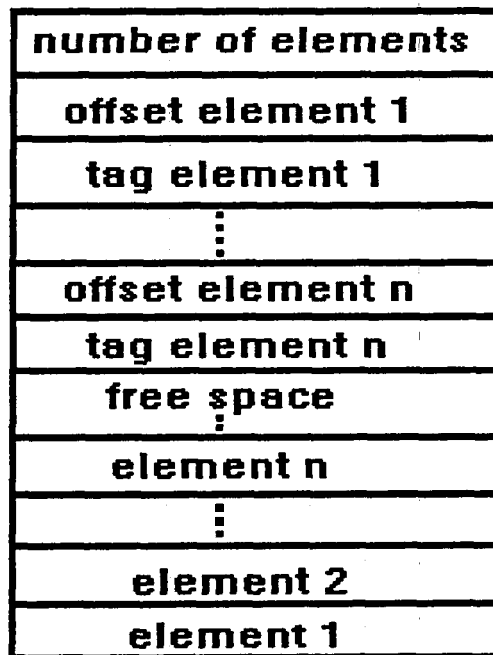
pFormList -> number of entries, offset element1, tag 1, . . . offset elementn, tag n

The element definitions are graphic primitives and groups of primitives known as objects.

A temporary group of primitives can be created by the form manipulation calls in order to perform operations on an area of the form. This special element is consumed by any subsequent form manipulation call.

Display List Memory Model

The display list will exist in a contiguous region of memory immediately after the form header. The form entries will begin at the lowest memory location and will grow towards high memory. The element definitions will begin at the high end of the region and grow towards the form.



Stroke font definitions will be stored in a stroke definition list immediately following the form display list.

Form Stroke List Definition

The forms resource supports user defined stroke font characters in the following manner:

Associated with each form is a stroke font definition list which contains the stroke characters defined for the form. This list is empty when a form is created, and can be added to through the use of `form_add_stroke`, and removed via `form_delete_stroke`.

DeskMate Technical Reference Forms Resource

In order to add a definition to the list, a structure containing the character definition must first be built. The format of a stroke character definition is as follows:

STROKE.size	int	- the size of the definition in bytes.
STROKE.ID.font#	char	- the font number of this character.
STROKE.ID.char	char	- the ASCII code for the character this character represents.

Immediately following the font# should be an array of characters which make up the definition of the character. All stroke characters are defined as a series of strokes on a 128 by 128 point grid with columns 1 thru 128 and rows 1 thru 128. Each stroke is defined by the coordinates of the polyline which make up the stroke. The value zero delineates strokes from each other.

For example:

The following is the stroke definition for a Roman character set capital A:

```
char ROMA[] = {
    30, 0, 2, 65          /* font 2 'A' */
    28, 100, 67, 16, 106, 100, 0, /* sides of A */
    67, 28, 100, 100, 0, /* thick side */
    39, 76, 89, 76, 0, /* cross bar */
    17, 100, 50, 100, 0, /* serif */
    83, 100, 117, 100 }; /* serif */
```

67, 28



28, 100 106, 100

The size of the entire definition is 30 bytes, the font number is 2, and the character is ASCII character 65 (which is 'A'). Each stroke is designated by the end points of the lines making up the stroke. Zeroes are used to indicate a 'pen-up' between individual strokes.

The calls listed below do not require a binding to the Forms Resource:

- form_display_region
- form_get_xy_extents
- form_scroll_down
- form_scroll_left
- form_scroll_right
- form_scroll_up
- form_update

The following binding calls allow for utilization of the forms resource:

- csr_form_bind_end
- csr_form_bind_init

DeskMate Technical Reference

Forms Resource

Structures/Defines

```
/*-----*/
/*
/* CSRFORM.H contains the Form structures and defines */
/*
/*-----*/

/* Forms Resource */
struct form_hdr_defn
{
    char    bNewList;        /* new list flag */
    char    bVideo;         /* video output flag */
    int     list_size;       /* size of form in words */
    int     stroke_size;    /* size of the stroke section in bytes */
    int     next_entry;     /* for internal use only */
    int     last_def;       /* for internal use only */
    int     tag_cnt;        /* for internal use only */
    int     *pGroup;        /* pointer to group header */
    int     *pFormList;     /* pointer to the form buffer */
    int     *pStrokes;      /* pointer to the stroke list */
};

typedef struct form_hdr_defn FORM_HDR;
```

form_hdr structures contain the information the forms resource needs to manipulate a given form. The application must possess one such structure for each form concurrently open. Multiple forms can be dealt with at any time, since each form possesses its own **FORM_HDR** structure and display list buffer. Each form's **FORM_HDR** structure contains the information the forms resource needs to manipulate the display list. The first step in building a form is to allocate a form header structure of type **FORM_HDR**.

FORM_HDR.bNewList

If set to zero, the forms resource will clear the display list buffer. If a previously created list is in the buffer, this should be set to 1 so the forms resource will calculate its internal pointers.

FORM_HDR.bVideo:

If set to 1 the forms resource will update the screen, if zero, no video output will take place. If set, a form window must be initialized before making any form manipulation calls.

FORM_HDR.list_size:

The size of the display list buffer in words.

FORM_HDR.stroke_size:

The size of the stroke definition list in bytes. This **MUST** be set to at least two bytes.

FORM_HDR.next_entry:

The offset in words from the start of the display list to where the next entry can be stored.

FORM_HDR.last_def:

The word offset to the last element definition stored in the display list.

FORM_HDR.tag_cnt:

Holds the last element tag assigned.

FORM_HDR.pGroup:

A pointer to the current element group. Zero if no valid group exists.

DeskMate Technical Reference

Forms Resource

FORM_HDR.pFormList:

A pointer to a contiguous region of memory where the display list will be built.

FORM_HDR.pStrokes:

A pointer to the stroke font definition buffer.

```
struct form_size_buf_defn
{
    int    list;        /* size of list */
    int    strokes;     /* size of strokes */
};
typedef struct form_size_buf_defn FORM_SIZE_BUF;

struct form_dst_defn
{
    int    x0;          /* x origin integer */
    int    x0f;         /* x origin fraction */
    int    y0;          /* y origin integer */
    int    y0f;         /* y origin fraction */
    int    x1;          /* x end integer */
    int    x1f;         /* x end fraction */
    int    y1;          /* y end integer */
    int    y1f;         /* y end fraction */
};
typedef struct form_dst_defn FORM_DST;
```

form_dst structures are used to pass coordinate information to the forms resource.

FORM_DST.x0:

The integer portion of the x axis world coordinate for the top left hand corner of the bounding rectangle.

FORM_DST.x0f:

The fractional portion of the x axis world coordinate for the top left hand corner of the bounding rectangle.

FORM_DST.y0:

The integer portion of the y axis world coordinate for the top left hand corner of the bounding rectangle.

FORM_DST.y0f:

The fractional portion of the y axis world coordinate for the top left hand corner of the bounding rectangle.

FORM_DST.x1:

The integer portion of the x axis world coordinate for the bottom right hand corner of the bounding rectangle.

FORM_DST.x1f:

The fractional portion of the x axis world coordinate for the bottom right hand corner of the bounding rectangle.

FORM_DST.y1:

The integer portion of the y axis world coordinate for the bottom right hand corner of the bounding rectangle.

DeskMate Technical Reference Forms Resource

FORM_DST.y1f:

The fractional portion of the y axis world coordinate for the bottom right hand corner of the bounding rectangle.

```
struct element_defn
{
    char    type;    /* element type */
    char    mod;     /* element attribute */
    int     x0;      /* x origin integer */
    int     x0f;     /* x origin fraction */
    int     y0;      /* y origin integer */
    int     y0f;     /* y origin fraction */
    int     x1;      /* x end integer */
    int     x1f;     /* x end fraction */
    int     y1;      /* y end integer */
    int     y1f;     /* y end fraction */
};
typedef struct element_defn ELEMENT;
```

When adding or replacing elements to a form it is necessary to construct the element definition within the proper structure type. All element, object, and group definitions, as well as CLIP_DRAW information on the clipboard, begin with data in a structure of type ELEMENT. When defining elements, the fractional portion of the element coordinates should be set to zero. The fractional portion is used internally when scaling elements in order to maintain accuracy. The x0, y0, x1, y1 define the bounding box for a group of elements.

ELEMENT.type:

One byte type identifiers: FORM_POLY_TYPE, FORM_ELLIP_TYPE, FORM_LINE_TYPE, FORM_BEV_TYPE, FORM_RECT_TYPE, FORM_TEXT_TYPE, FORM_OTHER. Further rendering of the element definition is totally dependent on this being correct.

FORM_POLY_TYPE	'P'
FORM_ELLIP_TYPE	'E'
FORM_LINE_TYPE	'L'
FORM_BEV_TYPE	'B'
FORM_RECT_TYPE	'R'
FORM_TEXT_TYPE	'T'
FORM_OTHER	'U'

ELEMENT.mod:

Used differently by each element type. See **form_add_element** for specific cases.

ELEMENT.x0:

The integer portion of the x axis world coordinate for the top left hand corner of the element's bounding rectangle.

ELEMENT.x0f:

The fractional portion of the x axis world coordinate for the top left hand corner of the element's bounding rectangle.

ELEMENT.y0:

The integer portion of the y axis world coordinate for the top left hand corner of the element's bounding rectangle.

ELEMENT.y0f:

DeskMate Technical Reference

Forms Resource

The fractional portion of the y axis world coordinate for the top left hand corner of the element's bounding rectangle.

ELEMENT.x1:

The integer portion of the x axis world coordinate for the bottom right hand corner of the element's bounding rectangle.

ELEMENT.x1f:

The fractional portion of the x axis world coordinate for the bottom right hand corner of the element's bounding rectangle.

ELEMENT.y1:

The integer portion of the y axis world coordinate for the bottom right hand corner of the element's bounding rectangle.

ELEMENT.y1f:

The fractional portion of the y axis world coordinate for the bottom right hand corner of the element's bounding rectangle.

```
FORM U ARC TYPE          'A'
FORM U IMAGE TYPE        'I'
FORM U POLYLINE TYPE      'P'

struct form_line_defn
{
    ELEMENT element;
    char    color;        /* line color */
    char    width;        /* line width */
};
typedef struct form_line_defn FORM_LINE;
```

In the special case of FORM_LINE elements, the coordinates specify the end points of the line rather than a bounding rectangle.

FORM_LINE.element:

The element definition.

FORM_LINE.element.type: FORM_LINE_TYPE.

FORM_LINE.element.mod: specifies the line style.

FORM_LINE.color:

The color of the line specified by COLOR1 - COLOR16.

FORM_LINE.width:

Line width defined type; LINE_WIDTH1 - LINE_WIDTH5.

```
struct form_rect_defn
{
    ELEMENT element;
    char    lncolor;      /* rectangle color */
    char    lnwidth;      /* rectangle width */
    char    lnstyle;      /* rectangle line style */
    char    bgnd_color;    /* background color of pattern */
    char    fgnd_color;    /* foreground color of pattern */
    char    pad;
};
typedef struct form_rect_defn FORM_RECT;
```

DeskMate Technical Reference Forms Resource

```
struct form_bev_defn
{
    ELEMENT element;
    char    lncolor;      /* rectangle color */
    char    lnwidth;      /* rectangle width */
    char    lnstyle;      /* rectangle line style */
    char    bgnd_color; /* background color of pattern */
    char    fgnd_color; /* foreground color of pattern */
    char    pad;
};
typedef struct form_bev_defn FORM_BEV;

struct form_ellipse_defn
{
    ELEMENT element;
    char    lncolor;      /* rectangle color */
    char    lnwidth;      /* rectangle width */
    char    lnstyle;      /* rectangle line style */
    char    bgnd_color; /* background color of pattern */
    char    fgnd_color; /* foreground color of pattern */
    char    pad;
};
typedef struct form_ellipse_defn FORM_ELLIPSE;
```

FORM_RECT, FORM_BEV, and FORM_ELLIPSE are similar structures containing the following variables where (XXXX) is RECT, BEV, or ELLIPSE.

FORM_(XXXX).element:

ELEMENT definition.

FORM_(XXXX).element.type: FORM_BEV_TYPE, FORM_ELLIPSE_TYPE, FORM_RECT_TYPE.

FORM_(XXXX).element.mod: specifies the fill pattern.

FORM_(XXXX).lncolor:

The color of the line as defined by: COLOR1 - COLOR16.

FORM_(XXXX).lnwidth:

Line width defined type: LINE_WIDTH1 - LINE_WIDTH5.

FORM_(XXXX).lnstyle:

Line type definition: LINE_SOLID, LINE_INVISIBLE, LINE_DOTTED, LINE_DASHED, LINE_DOT_DASHED, and LINE_DENSE_DOTTED.

FORM_(XXXX).fgnd_color:

Foreground color of the fill pattern as defined by: COLOR1 - COLOR16.

FORM_(XXXX).bgnd_color:

Background color of the fill pattern as defined by: COLOR1 - COLOR16.

FORM_(XXXX).pad:

Null.

```
struct form_text_defn
{
    ELEMENT element;
    char    attr;          /* character attribute for the string */
    char    color;         /* foreground color of string */
};
```

DeskMate Technical Reference

Forms Resource

```

char    nChars;      /* number of characters in the string */
char    rot;         /* character rotation */
char    *pString;    /* pointer to the string */
};
typedef struct form_text_defn FORM_TEXT;

```

FORM_TEXT.element:

ELEMENT definition.

FORM_TEXT.element.type: FORM_TEXT_TYPE.

FORM_TEXT.element.mod: specifies the font number.

FORM_TEXT.attr:

Character attribute defined type: NORMAL, BOLD, ITALIC, and UNDERLINE.

FORM_TEXT.color:

The text color as specified by: COLOR1 - COLOR16.

FORM_TEXT.nChars:

Number of characters in the string: 0 to 132.

FORM_TEXT.rot:

The orientation of the text string as defined by: FORM_ROT0, FORM_ROT90, FORM_ROT180, and FORM_ROT270. Rotation is ignored for system font.

```

FORM_ROT0      0
FORM_ROT90     1
FORM_ROT180    2
FORM_ROT270    3

```

FORM_TEXT.pString:

A pointer to an ASCII character string nChars long. Only printable characters should be in the string.

```

struct form_u_defn
{
    ELEMENT element;
    int      size;
};
typedef struct form_u_defn FORM_U;

struct form_gattr_defn
{
    char    line_fgnd_color;    /* line pattern foreground color */
    char    line_bgnd_color;    /* line pattern background color */
    char    line_style;        /* line style */
    char    line_width;        /* line width */
    char    pattern;           /* line pattern/fill pattern */
    char    bgnd_color;        /* fill pattern background color */
    char    fgnd_color;        /* fill pattern foreground color */
    char    pad;               /* padding */
};
typedef struct form_gattr_defn FORM_GATTR;

struct form_tattr_defn
{
    char    color;
    char    attr;
    char    rot;
    char    font;

```

Handwritten note: size of entire ^{element} form, including header, in words

DeskMate Technical Reference Forms Resource

```

};
typedef struct form_tattr_defn FORM_TATTR;

/* the ARC structure is from csrbase.h */
/* graphic primitive structures */
struct arc_defn
{
    int     base1_x;      /* first base point x */
    int     base1_y;      /* first base point y */
    int     base2_x;      /* second base point x */
    int     base2_y;      /* second base point y */
    int     apogee_x;     /* apogee x */
    int     apogee_y;     /* apogee y */
};
typedef struct arc_defn ARC;

struct form_u_arc_defn
{
    FORM_U   u header;      /* type U mod A */
    FORM_GATTR aAttr;
    int      xext;          /* Original x extent of bounding rect */
    int      yext;          /* Original y extent of bounding rect */
    ARC      arc;           /* arc definition */
};
typedef struct form_u_arc_defn FORM_U_ARC;

struct form_u_polyline_defn
{
    FORM_U   u header;      /* type U mod P */
    FORM_GATTR aAttr;
    int      xext;          /* original x extent of bounding rect */
    int      yext;          /* original y extent of bounding rect */
    char     bFill;         /* fill flag for polygon */
    char     bConnect;      /* 1st & last point connect flag */
    int      nPoints;       /* number of points in polygon */
};
typedef struct form_u_polyline_defn FORM_U_POLYLINE;

```

The points that make up the POLYLINE follow the FORM_U_POLYLINE structure. These points are relative to the origin of the element, **NOT** relative to the screen origin or active window.

The polyline is defined within a region with origin 0,0 and extents xorgext, yorgext. This definition will be rendered into the bounding rectangle of the ELEMENT when the element is displayed.

FORM_U_POLY.element.type char - FORM_POLY_TYPE.
 FORM_U_POLY.element.mod char - specifies the fill pattern.
 FORM_U_POLY.color: line color of the polyline.
 FORM_U_POLY.width: line width of the polyline.
 FORM_U_POLY.nPoints: length of pointlist in words.
 FORM_U_POLY.pPoints: pointers to array of points.

```

struct form_u_image_defn
{
    FORM_U   u header;      /* type U mod B */
    FORM_GATTR aAttr;
    char     bTransparent;  /* transparency flag for COLOR1 */
    int      bitmap_xext;   /* x extent of image in pels */
    int      bitmap_yext;   /* y extent of image in pels */
    char     nColors;       /* number of colors in image */
};

```

DeskMate Technical Reference

Forms Resource

```
typedef struct form_u_image_defn FORM_U_IMAGE;
```

FORM_U_IMAGE.u_header.FORM_U_element.type: FORM_U_IMAGE_TYPE.

FORM_U_IMAGE.u_header.FORM_U_element.mod: not used.

FORM_U_IMAGE.attr: not used.

FORM_U_IMAGE.pTransparent: indicates if the background will be solid.

FORM_U_IMAGE.bitmap_xext: pel width of bitmap, must fall on even byte boundary.

FORM_U_IMAGE.bitmap_yext: pel height of bitmap.

FORM_U_IMAGE.nColors: number of bits per pel. (1 for 2 colors, 2 for 4 colors and 4 for 16 colors)

The following is a sample which is an 8 * 8 rectangle which would be rendered into the rectangle from 100,200 to 1000,2000.

```
db      FORM_U_IMAGE_type      ;type
db      0                      ;mod
dw      100,0,200,0,1000,0,2000,0 ;bounding rectangle
dw      20                     ;size in words (from start of bounding rectangle)
db      0,0,0,0,0,0,0,0        ;attributes
db      Transparent            ;transparent background
dw      8                      ;x ext
dw      8                      ;y ext
db      1                      ;1 bit per pixel
db      11111111b              ;first row of definition
db      10000001b
db      10000001b
db      10000001b
db      10000001b
db      10000001b
db      10000001b
db      10000001b
db      11111111b              ;last row of definition
```

```
struct form_attr_defn
{
    char    text_color;
    char    text_attr;
    char    text_rot;
    char    font;
    char    line_fgnd_color;
    char    line_style;
    char    line_width;
    char    fill_pattern;
    char    bgnd_color;
    char    fgnd_color;
    char    bev_pad;
    char    line_bgnd_color;
    char    pad;
};
typedef struct form_attr_defn FORM_ATTR;
```

FORM_ATTR.text_color:

Color defined type: COLOR1 - COLOR16.

FORM_ATTR.text_attr:

Character attribute defined type: NORMAL, BOLD, ITALIC, UNDERLINE, INVERSE, GRAYED, and TRANSPARENT.

FORM_ATTR.text_rot:

DeskMate Technical Reference Forms Resource

Character rotation defined type: FORM_ROT0, FORM_ROT90, FORM_ROT180, and FORM_ROT270.

FORM_ATTR.font:

Font number: FORM_SYS_TEXT or 2 - 255.

FORM_SYS_TEXT - 1

FORM_ATTR.line_fgnd_color:

Color defined type: COLOR1 - COLOR16.

FORM_ATTR.line_style:

Line type definition: LINE_SOLID, LINE_INVISIBLE, LINE_DOTTED, LINE_DASHED, LINE_DOT_DASHED, and LINE_DENSE_DOTTED.

FORM_ATTR.line_width:

Line width defined type: LINE_WIDTH1 - LINE_WIDTH5.

FORM_ATTR.fill_pattern:

Fill pattern defined type: PATTERN1 - PATTERN20.

FORM_ATTR.fgnd_color:

Color defined type: COLOR1 - COLOR16.

FORM_ATTR.bgnd_color:

Color defined type: COLOR1 - COLOR16.

FORM_ATTR.bev_pad:

Rectangle bevel defined type: VID_NO_BEVEL, VID_BEVEL1, VID_BEVEL2, or VID_BEVEL3.

FORM_ATTR.line_bgnd_color:

Color defined type: COLOR1 - COLOR16.

FORM_ATTR.pad:

Null.

```
struct form_grp_defn
{
    ELEMENT element;
    int      num_parts;
    int      parts;
};
typedef struct form_grp_defn FORM_GRP;
```

DeskMate Technical Reference

Forms Resource

form_add_element (pFORM_HDR, pFORM_Element)
FORM_HDR *pFORM_HDR;
FORM_Element *pFORM_Element;

form_add_element adds the element specified by **pFORM_Element** to the indicated form. The supported elements are FORM_BEV, FORM_ELLIPSE, FORM_LINE, FORM_RECT, FORM_TEXT, FORM_U_ARC, FORM_U_IMAGE, and FORM_U_POLY.

Input Parameters

pFORM_HDR is a pointer to the form header structure of the form to be modified.
pFORM_Element points to a structure for one of the types listed above.

Return Value

<int>

The tag of the new element if successful.

DM_ERROR if insufficient buffer space remains to store the new element definition.

Special Notes

If a stroke font is specified, all stroke definitions must be added using **form_add_stroke**.

See Also

FORM_BEV structure.

FORM_ELLIPSE structure.

FORM_LINE structure.

FORM_RECT structure.

FORM_TEXT structure.

FORM_U_ARC structure.

FORM_U_IMAGE structure.

FORM_U_POLY structure.

form_add_stroke()

vid_put_image()

DeskMate Technical Reference Forms Resource

form_add_stroke (pFORM_HDR, pFORM_STROKE)
FORM_HDR *pFORM_HDR;
FORM_STROKE *pFORM_STROKE;

form_add_stroke adds the stroke definition to the form's stroke definition list.

Input Parameters

pFORM_HDR is a pointer to the form header structure of the form to be modified.
pFORM_STROKE is a pointer to the stroke font character definition for the new entry.

Return Value

<int>

DM_ERROR if insufficient space remains in the stroke definition list to store the new entry.

Special Notes

The stroke definitions should be in DeskMate format. The first word of the definition is the total size of the definition in bytes. The next byte of the definition is the ASCII code for the character which the definition represents. Following the ASCII code is a one byte font I.D. to identify the font style of the definition. The actual point by point stroke definition of the character follows. All points lie between 1 and 128, with zero specifying a pen up instruction. For instance, a simple A could be defined as follows:

15, 0, 65, 2, 1, 128, 64, 1, 128, 128, 0, 32, 64, 64, 64

DeskMate Technical Reference Forms Resource

form_break_object (pFORM_HDR, ElementTag)
FORM_HDR *pFORM_HDR;
int ElementTag;

form_break_object reduces the object into its constituent objects and elements by removing the pointer to the object from the form and inserting pointers to all of the object's parts.

Input Parameters

pFORM_HDR points to the form to be modified.
ElementTag should be an object tag.

Return Value

<int>

DM_ERROR if the indicated element is not an object or if insufficient space remains to break the object.

Special Notes

form_break_object deletes the object definition, and inserts pointers in the form list to its previous elements. If the element is not an object, nothing will happen. Tags for the objects parts can be located by using **form_find_element**.

DeskMate Technical Reference Forms Resource

form_clear (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_clear clears the indicated form's display list and stroke list.

Input Parameters

pFORM_HDR is a pointer to the form to be modified.

Return Value

None.

Special Notes

When the form is cleared, the memory allocated for the display list and the memory allocated for the stroke list will be initialized to all zeros.

Bugs

The actual size of the stroke list buffer within a form must be one byte greater than the byte specified for that buffer in the form header. **form_clear** will clear one byte too many based on the size indicated in the form header.

DeskMate Technical Reference Forms Resource

form_compress (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_compress compresses the display list to eliminate any free space between the form and element definitions. Updates the form header to reflect the new size.

Input Parameters

pFORM_HDR is a pointer to the form to be compressed.

Return Value

None.

Special Notes

Once a form has been compressed, no further calls which make additions can be made unless sufficient space is created by **form_delete**. **form_compress** destroys the current group pointer. **FORM_HDR.list_size** and **FORM_HDR.strokes_size** will be updated to their new values by **form_delete**. All form tags are invalidated by **form_compress**.

DeskMate Technical Reference Forms Resource

form_copy_element (pFORM_HDR, ElementTag)
FORM_HDR *pFORM_HDR;
int ElementTag;

form_copy_element copies the designated element onto the clipboard.

Input Parameters

pFORM_HDR points to the form to be copied from.
ElementTag is the tag of the element to be copied.

Return Value

<int>

DM_ERROR if the element definition is too large to fit on the clipboard.

Special Notes

The clipboard length will always be 2 bytes too small. To work around this the application must copy an extra two bytes when manually copying from the clipboard.

See Also

get_clipboard_info()
set_clipboard_info()

DeskMate Technical Reference Forms Resource

form_copy_group (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_copy_group copies the current group of elements onto the clipboard.

Input Parameters

pFORM_HDR is the form to be copied from.

Return Value

DM_ERROR if the group definition is too large to be copied onto the clipboard.

DeskMate Technical Reference

Forms Resource

form_cut_element (pFORM_HDR, ElementTag)
FORM_HDR *pFORM_HDR;
int ElementTag;

form_cut_element moves the element definition onto the clipboard and delete it from the form.

Input Parameters

pFORM_HDR points to the form to be cut from.
ElementTag is the tag of the element to be cut.

Return Value

<int>
DM_ERROR if the element is too large to fit on the form.

DeskMate Technical Reference Forms Resource

form_cut_group (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_cut_group moves the group of elements onto the clipboard and delete them from the form.

Input Parameters

pFORM_HDR is the form to be clipped from screen.

Return Value

DM_ERROR if the group will not fit on the clipboard.

DeskMate.Technical Reference Forms Resource

```
form_define_group ( pFORM_HDR, pFORM_DST )  
FORM_HDR *pFORM_HDR;  
FORM_DST *pFORM_DST;
```

form_define_group creates a temporary list from all elements of the form which are wholly contained within the specified bounding region.

Input Parameters

pFORM_HDR is a pointer to the form to be accessed.

pFORM_DST points to the structure which holds (x0, y0, x1, y1) which define the bounding rectangle of the group. Only elements totally within the region are included.

Return Value

<int>

The number of elements in the group.

DM_ERROR if insufficient space remains in the form to create a group.

NULL if no elements can be found in the region.

Special Notes

The application may specify a temporary group of elements on which some operation may be performed. Only one group exists at a time, and will be destroyed if any elements operations are performed.

The coordinates of the group bounding box *must* be specified with x0,y0 as the top left hand corner of the region and x1,y1 as the lower right hand corner.

The bounding rectangle of the group will be set to the minimum region which encompasses its elements.

DeskMate Technical Reference Forms Resource

form_delete_element (pFORM_HDR, ElementTag)
FORM_HDR *pFORM_HDR;
int ElementTag;

form_delete_element removes the element definition from the form.

Input Parameters

pFORM_HDR is a pointer to the form to be modified.
ElementTag defines the element to be deleted.

Return Value

<int>
DM_ERROR if an invalid element tag is specified.

Special Notes

form_delete_element removes the element definition from the form. If the element is an object, all elements of the object will be deleted. The tag for the deleted element will no longer be valid.

DeskMate Technical Reference Forms Resource

form_delete_group (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_delete_group deletes all group elements from the form. Destroys the group element.

Input Parameters

pFORM_HDR points to the form to be modified.

Return Value

<int>

DM_ERROR if no group exists.

Special Notes

Operates exactly as **form_delete_element** with the group treated as an object.

DeskMate Technical Reference Forms Resource

form_delete_stroke (pFORM_HDR, StrokeID)
FORM_HDR *pFORM_HDR;
int StrokeID;

form_delete_stroke deletes the definition for the indicated stroke font character from the form's stroke definition list.

Input Parameters

pFORM_HDR is a pointer to the to be modified.

StrokeID is the identifier of the stroke character definition to be removed.

Return Value

<int>

DM_ERROR if an invalid **StrokeID** is specified.

Special Notes

The **StrokeID**'s are of the form ASCII character, font number. For instance, the character A of font three (3) would be identified by 4103 hex.

DeskMate Technical Reference Forms Resource

form_delete_unused_strokes (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_delete_unused_strokes deletes any of the form's stroke definitions which are not required by existing form text element definitions.

Input Parameters

pFORM_HDR points to the form to be modified.

Return Value

None.

Special Notes

form_delete_unused_strokes examines all of the form's text elements to determine which stroke definitions are used. It then deletes the unused definitions to free space in the stroke list buffer.

DeskMate Technical Reference Forms Resource

form_display_region (pFORM_HDR, pMAPRECT)
FORM_HDR *pFORM_HDR;
MAPRECT *pMAPRECT;

form_display_region displays any of the form's elements which intersect the specified region.

Input Parameters

pFORM_HDR points to the form to be displayed.

pMAPRECT points to a MAPRECT structure which holds the coordinates of the points that define the region to be displayed.

Return Value

None.

Special Notes

xorg holds x0, xext holds x1, yorg holds y0, yext holds y1.

form_display_region does not require loading of the forms resource.

DeskMate Technical Reference Forms Resource

form_dup_element (pFORM_HDR, ElementTag, pFORM_DST)
FORM_HDR *pFORM_HDR;
int ElementTag;
FORM_DST *pFORM_DST;

form_dup_element makes a scaled copy of the element in the specified bounding rectangle.

Input Parameters

pFORM_HDR is a pointer to the form to be modified.

ElementTag is the element tag.

pFORM_DST is a pointer to a structure which contains the (x0, y0) and (x1, y1) to define the bounding box to move the element into.

Return Value

<int>

The tag of the new element definition.

DM_ERROR is returned if insufficient space exist to copy the element or an invalid tag is specified.

Special Notes

form_dup_element creates new element definitions, and inserts pointers in the form list. If the element is an object, all elements of the object will be affected.

Except for the special case of a line element, the destination coordinates must be specified with x0,y0 as the top left hand corner of the region and x1,y1 as the lower right hand corner.

DeskMate Technical Reference Forms Resource

form_dup_group (pFORM_HDR, pFORM_DST)
FORM_HDR *pFORM_HDR;
FORM_DST *pFORM_DST;

form_dup_group creates a scaled copy of the group in the bounding box. Destroys the old group, replacing it with a new group containing the new elements.

Input Parameters

pFORM_HDR points to the form to be modified.

pFORM_DST points to the structure which contains the (x0, y0) and (x1, y1) which define the area to copy the group into.

Return Value

<int>

DM_ERROR if no group exists.

Special Notes

Operates exactly as **form_copy_element** with the group treated as an object.

DeskMate Technical Reference Forms Resource

form_element_to_bottom (pFORM_HDR, ElementTag)
FORM_HDR *pFORM_HDR;
int ElementTag;

form_element_to_bottom moves the element definition to the bottom of the element definitions so that the element will be drawn "under" the other elements.

Input Parameters

pFORM_HDR points to the form to be modified.
ElementTag is the tag.

Return Value

<int>
DM_ERROR if an invalid element tag is specified.

Special Notes

form_element_to_bottom shifts all element definitions up by the size of the moved element and then inserts the element definition under all other definitions. If the element is an object, all elements of the object will be affected. **form_element_to_bottom** will alter all element offsets, but will not affect element tags.

DeskMate Technical Reference Forms Resource

form_element_to_top (pFORM_HDR, ElementTag)
FORM_HDR *pFORM_HDR;
int ElementTag;

form_element_to_top moves the element definition to the top of the element definitions so that the element will be drawn "over" the other elements.

Input Parameters

pFORM_HDR points to the form to be modified.
ElementTag is a pointer to the element.

Return Value

<int>
DM_ERROR if an invalid element tag is specified.

Special Notes

form_element_to_top performs the equivalent actions of **copy_element** followed by deleting the original. If the element is an object, all elements of the object will be affected.

DeskMate Technical Reference

Forms Resource

form_expand (pFORM_HDR, HowMuchList, HowMuchStroke)
FORM_HDR *pFORM_HDR;
int HowMuchList, HowMuchStroke;

form_expand expands the form display list to create free space between the form and element definitions, and adds additional room to the stroke list.

Input Parameters

pFORM_HDR is a pointer to the form to be modified.

HowMuchList is an integer indication of the number of additional bytes allocated for the display list.

HowMuchStroke is an integer indication of the number of additional bytes allocated for the stroke list.

Return Value

None.

Special Notes

form_expand destroys the current group pointer.

DeskMate Technical Reference

Forms Resource

form_find_element (pFORM_HDR, x, y, n)
FORM_HDR *pFORM_HDR;
int x, y, n;

form_find_element searches for elements (an object is also an element) which contain the point (x, y) within their bounding region. Returns a pointer to the nth satisfying element in the display list.

Input Parameters

pFORM_HDR points to the form to be searched.
n is which satisfying element to be returned.
x, y is the point from which to search.

Return Value

<int>

The tag of the nth satisfying element.
NULL if no element satisfies the search.

Special Notes

form_find_element examines the bounding boxes of the element definitions. If the element is an object, only the bounding box of the object will be considered.

DeskMate Technical Reference Forms Resource

form_find_stroke (pFORM_HDR, StrokeID)
FORM_HDR *pFORM_HDR;
int StrokeID;

form_find_stroke returns a pointer to the stroke character definition stored in the form stroke list.

Input Parameters

pFORM_HDR is a pointer to the Forms header structure.
StrokeID is the stroke definition identifier.

Return Value

A pointer to the corresponding stroke definition.
DM_ERROR if the **StrokeID** can't be matched.

Special Notes

Generally the **StrokeID** should consist of a one byte ASCII value and the font number.

DeskMate Technical Reference Forms Resource

form_get_pointer (pFORM_HDR, ElementTag)
FORM_HDR *pFORM_HDR;
int ElementTag;

form_get_pointer returns a pointer to the beginning of the element definition.

Input Parameters

pFORM_HDR is a pointer to the form header structure of the form to be modified.
ElementTag is the tag of the element to be located.

Return Value

A pointer to the element definition. The first byte of the definition indicates the definition type.
DM_ERROR will be returned if an invalid tag is specified.

Special Notes

The element definitions are stored exactly as the structures used to specify them with the following exception. Variable length definitions such as text and polyline do not hold a pointer to their elements, instead their elements are stored beginning contiguously with the definition, beginning at the location which would have been the element array pointer.

DeskMate Technical Reference Forms Resource

form_get_xy_extents (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_get_xy_extents examines the bounding rectangles of all of the elements in a form, and returns the maximum x and y values.

Input Parameters

pFORM_HDR is a pointer to the form to be examined.

Return Value

<long integer>

The most significant word is the x extent. The least significant word is the y extent.

Special Notes

The form should be opened before calling **form_get_xy_extents**.

form_get_xy_extents does not require loading of the form resource.

DeskMate Technical Reference Forms Resource

form_group_to_bottom (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_group_to_bottom moves the element definitions of the group to the bottom of the element definitions so that the elements will be drawn "under" the other elements.

Input Parameters

pFORM_HDR is a pointer to the form to be modified.

Return Value

<int>

DM_ERROR if no group exists.

Special Notes

form_group_to_bottom shifts all element definitions up by the size of the moved elements and then inserts the group element definitions under all other definitions. If the any element is an object, all elements of the object will be affected. **form_group_to_bottom** will alter all element pointers.

DeskMate Technical Reference Forms Resource

form_group_to_top (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_group_to_top moves the element definitions of the group to the top of the element definitions so that the elements will be drawn "over" the other elements.

Input Parameters

pFORM_HDR points to the form to be modified.

Return Value

<int>

DM_ERROR if no group exists.

Special Notes

form_group_to_top performs the equivalent actions of a **copy_group** followed by deleting the original. If the element is an object, all elements of the object will be affected. **form_group_to_top** will alter all element pointers above the original group element definitions.

DeskMate Technical Reference

Forms Resource

form_make_object (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_make_object creates an object from all of the elements in the group. Removes all tags to the individual elements from the form, replacing them with a single pointer to the object element. Destroys the group element.

Input Parameters

pFORM_HDR points to the form to be modified.

Return Value

<int>

The tag for the new element definition.

DM_ERROR is returned if no valid group exists.

Special Notes

Converts the group definition to an object definition. Inserts a pointer to the new object into the form. Deletes any references to the elements of the new object.

DeskMate Technical Reference Forms Resource

form_mod_attr (pFORM_HDR, ElementTag, pFORM_ATTR)
FORM_HDR *pFORM_HDR;
int ElementTag;
FORM_ATTR *pFORM_ATTR;

form_mod_attr modifies the specified attributes of the element.

Input Parameters

pFORM_HDR is a pointer to the form to be modified.

ElementTag is an element tag.

pFORM_ATTR points to the structure which contains the new attributes. Any attributes which have the value **DM_ERROR** will be unchanged.

Return Value

<int>

DM_ERROR if the element definition structure type does not match the element type.

Special Notes

form_mod_attr modifies the attributes of the elements definitions, but does not have to change the pointers in the form list. If the element is an object, all elements of the object which are of the element definition type will be affected.

DeskMate Technical Reference Forms Resource

form_mod_grp_attr (pFORM_HDR, pFORM_Element)
FORM_HDR *pFORM_HDR;
FORM_Element *pFORM_Element;

form_mod_grp_attr modifies the specified attributes of the element within the group.

Input Parameters

pFORM_HDR is a pointer to the form to be modified.

pFORM_Element points to the element definition structure of type **FORM_BEV**, **FORM_ELLIPSE**, **FORM_LINE**, **FORM_RECT**, **FORM_TEXT**, **FORM_U_ARC**, **FORM_U_IMAGE**, or **FORM_U_POLY**, which contains the new attributes. Any attributes of the element definition which contain **DM_ERROR** will be unchanged.

Return Value

<int>

DM_ERROR if the element definition structure type does not match any of the element types in the group.

Special Notes

form_mod_grp_attr modifies the attributes of the group element definitions which are of the same type as the element definition. It does not have to change the pointers in the form list.

See Also

FORM_BEV structure.
FORM_ELLIPSE structure.
FORM_LINE structure.
FORM_RECT structure.
FORM_TEXT structure.
FORM_U_ARC structure.
FORM_U_IMAGE structure.
FORM_U_POLY structure.

DeskMate Technical Reference Forms Resource

form_move_element (pFORM_HDR, ElementTag, pFORM_DST)
FORM_HDR *pFORM_HDR;
int ElementTag;
FORM_DST *pFORM_DST;

form_move_element moves and re-sizes the element into the new bounding rectangle. Updates the coordinate values of the element.

Input Parameters

pFORM_HDR points to the form to be modified.

ElementTag is the element tag.

pFORM_DST points to the destination structure which contains the coordinates (x0, y0) and (x1,y1) to define the bounding box to move the element into.

Return Value

<int>

DM_ERROR if an invalid tag is specified.

Special Notes

form_move_element modifies the coordinates of the elements definitions, but does not have to change the pointers in the form list. If the element is an object, all elements of the object will be affected.

Except for the special case of a line element, the destination coordinates must be specified with x0,y0 as the top left hand corner of the rectangle and x1, y1 as the lower right corner.

DeskMate Technical Reference Forms Resource

form_move_group (pFORM_HDR, pFORM_DST)
FORM_HDR *pFORM_HDR;
FORM_DST *p_destination;

form_move_group moves and re-sizes the group into the new bounding box. Updates the coordinate values of all of the group elements.

Input Parameters

pFORM_HDR is a pointer to the form to be modified.
pFORM_DST points to the structure which holds (x0, y0) and (x1, y1) which define the area to move the group into.

Return Value

<int>
DM_ERROR if no group exists.

Special Notes

Operates exactly as **form_move_element** with the group treated as an object.
x0,y0 must be the top left hand corner of the destination region, x1,y1 must be the lower right hand region.

DeskMate Technical Reference Forms Resource

form_open (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_open initializes the indicated form buffer. It sets initial pointers to the element definition and the stroke character definitions.

Input Parameters

pFORM_HDR pointer to the header of the buffer for the list.

Return Value

None.

Special Notes

Prior to issuing **form_open**, the caller must create a form header structure for the form. In this structure the following must be set by the user:

FORM_HDR.bNewList:

If this is set to zero, the forms resource will clear the display list buffer. If a previously created list is in the buffer, this should be set to 1 so the forms resource will calculate its internal pointers.

FORM_HDR.bVideo:

If this is 1 the forms resource will update the screen, if zero, no video output will take place.

FORM_HDR.list_size:

The size of the display list buffer in words.

FORM_HDR.stroke_size:

The size of the stroke list.

The remainder of the FORM_HDR elements will either be calculated by the forms resource or set by subsequent calls. These elements may be read, but should not be written to by the caller.

When the form is opened, the memory allocated may already contain a pre-created form display list. If so, the routine must determine the bottom of the form list and the top of the element definitions. If no display list is present, the buffer is initialized to all zeros.

WARNING: DO NOT MODIFY THE FOLLOWING FORM_HDR STRUCTURE ELEMENTS:

next_entry

last_def

pGroup

pFormList

pStrokes

Bugs

The actual size of the stroke list buffer within a form must be one byte greater than the byte specified for that buffer in the form header. **form_open** will clear one byte too many based on the size indicated in the form header.

DeskMate Technical Reference Forms Resource

form_paste (pFORM_HDR)
FORM_HDR *pFORM_HDR;

form_paste adds the clipboard display list to the form.

Input Parameters

pFORM_HDR is a pointer to the form to be modified.

Return Value

<int>

The tag of the object which contains the elements from the clipboard.

Six (6) if non-form data is in the clipboard, the tag will not be incremented.

DM_ERROR if insufficient space.

Special Notes

The elements pasted from the clipboard will be contained in an object.

DeskMate Technical Reference Forms Resource

form_region_empty (pFORM_HDR, pMAPRECT)
FORM_HDR *pFORM_HDR;
MAPRECT *pMAPRECT;

form_region_empty determines if the form contains any elements which intersect the indicated region.

Input Parameters

pFORM_HDR is a pointer to the form header structure of the form to be modified.

pMAPRECT is a pointer to a MAPRECT structure which contains the coordinates of the area to be searched.

Return Value

NULL if the region is empty.

One (1) if the form intersects the region specified by **pMAPRECT**.

DeskMate Technical Reference

Forms Resource

form_replace_element (pFORM_HDR, ElementTag, pFORM_Element)
FORM_HDR *pFORM_HDR;
int ElementTag;
FORM_Element *pFORM_Element;

form_replace_element replaces the definition of the element specified by **ElementTag** with the definition pointed to by **pFORM_Element**. The new definition will be rendered in the same order as the replaced element.

Input Parameters

pFORM_HDR is a pointer to the form header structure of the form to be modified.

ElementTag is the tag of the element to be replaced.

pFORM_Element points to an element definition structure of type **FORM_BEV**, **FORM_ELLIPSE**, **FORM_LINE**, **FORM_RECT**, **FORM_TEXT**, **FORM_U_ARC**, **FORM_U_IMAGE**, or **FORM_U_POLY**.

Return Value

DM_ERROR if the **ElementTag** specified is not implemented or insufficient buffer space remains to store the new element definition.

Special Notes

form_replace_element is used to preserve rendering order. A typical use is when replacing one text string with another.

See Also

FORM_BEV structure.

FORM_ELLIPSE structure.

FORM_LINE structure.

FORM_RECT structure.

FORM_TEXT structure.

FORM_U_ARC structure.

FORM_U_IMAGE structure.

FORM_U_POLY structure.

DeskMate Technical Reference

Forms Resource

form_scroll_down (pFORM_HDR, WorldCordY)
FORM_HDR *pFORM_HDR;
int WorldCordY;

form_scroll_down scrolls the indicated form down by the indicated number of world coordinates. The form will scroll within the window.

Input Parameters

pFORM_HDR is a pointer to the header structure of the desired form.
WorldCordY is the number of world coordinates to scroll the form.

Return Value

None.

Special Notes

form_scroll_down activates the form's window, scrolls the information within that window, and then redraws the new data into the window. It then reactivates the caller's window.
form_scroll_down does not require loading of the forms resource.

DeskMate Technical Reference

Forms Resource

form_scroll_left (pFORM_HDR, WorldCordX)
FORM_HDR *pFORM_HDR;
int WorldCordX;

form_scroll_left scrolls the indicated form to the left by the indicated number of world coordinates. The form will scroll within the window.

Input Parameters

pFORM_HDR is a pointer to the header structure of the desired form.
WorldCordX is the number of world coordinates to scroll the form.

Return Value

None.

Special Notes

form_scroll_left activates the form's window, scrolls the information within that window, and then redraws the new data into the window. It then reactivates the caller's window.
form_scroll_left does not require loading of the forms resource.

DeskMate Technical Reference

Forms Resource

form_scroll_right (pFORM_HDR, WorldCordX)
FORM_HDR *pFORM_HDR;
int WorldCordX;

form_scroll_right scrolls the indicated form to the right by the indicated number of world coordinates. The form will scroll within the window.

Input Parameters

pFORM_HDR is a pointer to the header structure of the desired form.
WorldCordX is the number of world coordinates to scroll the form.

Return Value

None.

Special Notes

form_scroll_right activates the form's window, scrolls the information within that window, and then redraws the new data into the window. It then reactivates the caller's window.
form_scroll_right does not require loading of the forms resource.

DeskMate Technical Reference

Forms Resource

form_scroll_up (pFORM_HDR, WorldCordY)
FORM_HDR *pFORM_HDR;
int WorldCordY;

form_scroll_up scrolls the indicated form up by the indicated number of world coordinates. The form will scroll within the window.

Input Parameters

pFORM_HDR is a pointer to the header structure of the desired form.
WorldCordY is the number of world coordinates to scroll the form.

Return Value

None.

Special Notes

form_scroll_up activates the form's window, scrolls the information within that window, and then redraws the new data into the window. It then reactivates the caller's window.
form_scroll_up does not require loading of the forms resource.

DeskMate Technical Reference Forms Resource

1989 DeskMate 03.03.0x ONLY

form_specify_group (pFORM_HDR, pTagList, nTags)
FORM_HDR *pFORM_HDR;
int *pTagList;
int nTags;

form_specify_group makes the list of element tags into a group.

Input Parameters

pFORM_HDR is a pointer to the header structure of the desired form.

pTagList is a pointer to an integer tag list.

nTags is the number of tags pointed to by **pTagList**.

Return Value

DM_ERROR if the group would not fit.

DeskMate Technical Reference

Forms Resource

form_update (pFORM_HDR, ClearScreen)
FORM_HDR *pFORM_HDR;
int ClearScreen

form_update draws (or redraws) the current contents of the form into the currently defined form window.

Input Parameters

pFORM_HDR is a pointer to the form to be displayed.

ClearScreen indicates whether or not the window will be cleared prior to update. If **ClearScreen** is **ENABLED** the screen will be cleared; if **DISABLED** the screen will not be cleared.

Return Value

None.

Special Notes

form_update does not require loading of the forms resource.

**DeskMate Technical Reference
Forms Resource**

This page intentionally left blank

These calls are available in the
3.03 SDK under different names.
Change 'e' to 'f', viz.:

`eform_bind_init()`

→ `fform_bind_init()`

Far Form Resource

Table of Contents

Far Form Resource General Description/Notes	12-55
eform_add_element ..Adds a specified element to the form.....	12-56
eform_add_stroke ...Adds a stroke definition to the form.....	12-57
eform_bind_endTerminates forms resource loaded.....	12-58
eform_bind_initBinds to far forms resource.....	12-59
eform_break_object .Reduces one object to constituent objects..	12-60
eform_calc_form_size .Calculates memory needed for form.....	12-61
eform_clearClears the form's display and stroke lists.	12-62
eform_compressCompress free space from the form.....	12-63
eform_copy_element .Copies element to the clipboard.....	12-64
eform_copy_group ...Copies group of elements to clipboard.....	12-65
eform_cut_element ..Cuts the form element to the clipboard.....	12-66
eform_cut_groupCuts a group of elements to clipboard.....	12-67
eform_define_group .Defines a group of elements for the form...	12-68
eform_delete_element .Deletes element definition from form.....	12-69
eform_delete_group ...Deletes all group elements from form.....	12-70
eform_delete_stroke ..Deletes stroke character from form.....	12-71
eform_delete_unused_strokes .Deletes non-required strokes.....	12-72
eform_display_region .Displays form elements in a region.....	12-73
eform_dup_elementCreates a scaled copy of an element.....	12-74
eform_dup_groupCreates a scaled copy of a group.....	12-75
eform_element_to_bottomMoves element to bottom of defs....	12-76
eform_element_to_topMoves element to top of defs.....	12-77
eform_expandExpands the form display list.....	12-78
eform_find_element .Finds an element in a rectangle.....	12-79
eform_find_stroke ..Finds a stroke character.....	12-80
eform_get_extents ..Gets maximum x,y values for all elements...	12-81
eform_get_pointer ..Gets a pointer to element definition.....	12-82
eform_group_to_bottomMoves group to bottom of defs.....	12-83
eform_group_to_topMoves group to top of defs.....	12-84
eform_make_object ..Makes an object from elements in group....	12-85
eform_mod_attrModifies element attributes.....	12-86
eform_mod_grp_attr .Modifies element attributes in the group...	12-87

eform_move_element .Moves and resizes an element.....12-88
eform_move_group ...Moves and resizes a group of elements.....12-89
eform_openInitializes the form buffer.....12-90
eform_pasteAdds the clipboard display to the form.....12-91
eform_region_empty .Determines if the form contains elements...12-92
eform_replace_elementReplaces an element definition.....12-93
eform_scroll_down ..Scrolls the form down.....12-94
eform_scroll_left ..Scrolls the form to the left.....12-95
eform_scroll_right .Scrolls the form to the right.....12-96
eform_scroll_upScrolls the form up.....12-97
eform_specify_group.Makes list of element tags into a group....12-98
eform_updateDraws/redraws the current form.....12-99

DeskMate Technical Reference Far Forms Resource

General Description/Notes

The calls in this section are only available when using 1989 DeskMate 03.03.0x or later.

For calling the enhanced forms resource routines require a prefix of "e" on each of the form calls and to provide a far pointer to the `FORM_HDR` structure.

The Enhanced Form Manager will contain all of the functionality of the original form manager as well as font support. The near form calls are still supported, using the `csr_bind_init`, however they do not support fonts. The eform calls expect all pointer parameters to be far.

See the beginning of this section for a more complete General Description/Notes section. The structures and defines are the same as those used in the forms resource at the beginning of this section.

In order for the application to use the calls listed in this section, it must make a binding call, `eform_bind_init` and `eform_bind_end`. See the Desk Executive section for a list of all of the DeskMate initialization calls. The enhanced forms resource initialization calls are listed here for your convenience.

The `eform_bind_init` call contains a parameter that allows the font support to be turned off. If the flag is set to `NULL` the form manager will not be able to support fonts. If the flag is set to `TRUE`, then fonts are supported and applications will also have the ability to access the font engine or the enhanced video driver directly. See the Font Manager section for a list of Font calls.

The calls listed below do not require a binding to the Enhanced (Far) Forms Resource:

- `eform_display_region`
- `eform_get_extents`
- `eform_scroll_down`
- `eform_scroll_left`
- `eform_scroll_right`
- `eform_scroll_up`
- `eform_update`

DeskMate Technical Reference Far Forms Resource

eform_add_element (fpFORM_HDR, fpFORM_Element)
FORM_HDR far *fpFORM_HDR;
FORM_Element far *fpFORM_Element;

eform_add_element adds the element specified by **pFORM_Element** to the indicated form. The supported elements are **FORM_BEV**, **FORM_ELLIPSE**, **FORM_LINE**, **FORM_RECT**, **FORM_TEXT**, **FORM_U_ARC**, **FORM_U_IMAGE**, and **FORM_U_POLY**.

Input Parameters

fpFORM_HDR is a far pointer to the form header structure of the form to be modified.
fpFORM_Element is a far points to a structure for one of the types listed above.

Return Value

<int>

The tag of the new element if successful.

DM_ERROR if insufficient buffer space remains to store the new element definition.

Special Notes

If a stroke font is specified, all stroke definitions must be added using **form_add_stroke**.

See Also

FORM_BEV structure.
FORM_ELLIPSE structure.
FORM_LINE structure.
FORM_RECT structure.
FORM_TEXT structure.
FORM_U_ARC structure.
FORM_U_IMAGE structure.
FORM_U_POLY structure.
form_add_stroke()
vid_put_image()

DeskMate Technical Reference Far Forms Resource

eform_add_stroke (fpFORM_HDR, fpFORM_STROKE)
FORM_HDR far *fpFORM_HDR;
FORM_STROKE far *fpFORM_STROKE;

eform_add_stroke adds the stroke definition to the form's stroke definition list.

Input Parameters

fpFORM_HDR is a far pointer to the form header structure of the form to be modified.
fpFORM_STROKE is a far pointer to the stroke font character definition for the new entry.

Return Value

<int>

DM_ERROR if insufficient space remains in the stroke definition list to store the new entry.

Special Notes

The stroke definitions should be in DeskMate format. The first word of the definition is the total size of the definition in bytes. The next byte of the definition is the ASCII code for the character which the definition represents. Following the ASCII code is a one byte font I.D. to identify the font style of the definition. The actual point by point stroke definition of the character follows. All points lie between 1 and 128, with zero specifying a pen up instruction. For instance, a simple A could be defined as follows:

15, 0, 65, 2, 1, 128, 64, 1, 128, 128, 0, 32, 64, 64, 64

DeskMate Technical Reference Far Forms Resource

eform_bind_end ()

eform_bind_end terminates the forms resource loaded by **eform_bind_init** and unbinds any resources loaded by the enhanced (far) forms resource.

Input Parameters

None.

Return Value

None.

Special Notes

eform_bind_end requires the application to link with DeskMate 03.05 or later libraries (DM.LIB or DMMED.LIB).

DeskMate Technical Reference Far Forms Resource

eform_bind_init (Font)

eform_bind_init loads and links to the forms resource as an enhanced (far) forms resource. This functionality is only available in for use in DeskMate 03.05 and later.

Input Parameters

<char>

If **Font** is **FALSE** the enhanced form manager will not support fonts. The font resource and enhanced video driver will not be loaded into memory.

If **Font** is **TRUE** the enhanced form manager will support fonts, and the enhanced video driver will be loaded into memory.

Return Value

<int>

DM_OK if everything was loaded correctly.

DM_ERROR if the forms resource could not be loaded.

Special Notes

eform_bind_init requires the application to link with DeskMate 03.05 or later libraries (DM.LIB or DMMED.LIB).

DeskMate Technical Reference Far Forms Resource

eform_break_object (fpFORM_HDR, ObjectTag)
FORM_HDR far *fpFORM_HDR;
int ObjectTag;

eform_break_object reduces the object into its constituent objects and elements by removing the pointer to the object from the form and inserting pointers to all of the object's parts.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.
ObjectTag should be an object tag.

Return Value

<int>

DM_ERROR if the indicated element is not an object or if insufficient space remains to break the object.

Special Notes

eform_break_object deletes the object definition, and inserts pointers in the form list to its previous elements. If the element is not an object, nothing will happen. Tags for the objects parts can be located by using **eform_find_element**.

DeskMate Technical Reference Far Forms Resource

eform_calc_form_size (int DisplayList, int StrokeList)

eform_calc_form_size calculates the amount of memory needed to create the form. The form must not exceed 64K.

Input Parameters

DisplayList determines the amount of memory allocated for elements.

StrokeList determine the amount of memory allocated for stroke fonts. If stroke fonts are not used, **StrokeList** should be set to zero.

Return Value

Size of form required (in bytes).

DeskMate Technical Reference Far Forms Resource

eform_clear (fpFORM_HDR)
FORM_HDR far *fpFORM_HDR;

eform_clear clears the indicated form's display list and stroke list.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

Return Value

None.

Special Notes

When the form is cleared, the memory allocated for the display list and the memory allocated for the stroke list will be initialized to all zeros.

DeskMate Technical Reference

Far Forms Resource

eform_compress (fpFORM_HDR)
FORM_HDR far *fpFORM_HDR;

eform_compress compresses the display list to eliminate any free space between the form and element definitions. Updates the form header to reflect the new size.

Input Parameters

fpFORM_HDR is a far pointer to the form to be compressed.

Return Value

None.

Special Notes

Once a form has been compressed, no further calls which make additions can be made unless sufficient space is created by **eform_delete**. **eform_compress** destroys the current group pointer.

FORM_HDR.list_size and FORM_HDR.strokes_size will be updated to their new values by **eform_delete**. All form tags are invalidated by **eform_compress**.

DeskMate Technical Reference Far Forms Resource

eform_copy_element (pFORM_HDR, ElementTag)
FORM_HDR far *fpFORM_HDR;
int ElementTag;

eform_copy_element copies the designated element onto the clipboard.

Input Parameters

fpFORM_HDR is a far pointer to the form to be copied from.
ElementTag is the tag of the element to be copied.

Return Value

<int>

DM_ERROR if the element definition is too large to fit on the clipboard.

Special Notes

The clipboard length will always be 2 bytes too small. To work around this the application must copy an extra two bytes when manually copying from the clipboard.

See Also

get_clipboard_info()
set_clipboard_info()

DeskMate Technical Reference Far Forms Resource

eform_copy_group (fpFORM_HDR)
FORM_HDR far *fpFORM_HDR;

eform_copy_group copies the current group of elements onto the clipboard.

Input Parameters

fpFORM_HDR is a far pointer to the form to be copied from.

Return Value

DM_ERROR if the group definition is too large to be copied onto the clipboard.

DeskMate Technical Reference Far Forms Resource

eform_cut_element (fpFORM_HDR, ElementTag)
FORM_HDR far *fpFORM_HDR;
int ElementTag;

eform_cut_element moves the element definition onto the clipboard and delete it from the form.

Input Parameters

fpFORM_HDR is a far pointer to the form to be cut from.
ElementTag is the tag of the element to be cut.

Return Value

<int>
DM_ERROR if the element is too large to fit on the form.

DeskMate Technical Reference Far Forms Resource

eform_cut_group (fpFORM_HDR)
FORM_HDR far *fpFORM_HDR;

eform_cut_group moves the group of elements onto the clipboard and delete them from the form.

Input Parameters

fpFORM_HDR is a far pointer to the form to be clipped from screen.

Return Value

DM_ERROR if the group will not fit on the clipboard.

DeskMate Technical Reference Far Forms Resource

eform_define_group (fpFORM_HDR, fpFORM_DST)
FORM_HDR far *fpFORM_HDR;
FORM_DST far *fpFORM_DST;

eform_define_group creates a temporary list from all elements of the form which are wholly contained within the specified bounding region.

Input Parameters

fpFORM_HDR is a far pointer to the form to be accessed.

fpFORM_DST is a far pointer to the structure which holds (x0, y0, x1, y1) which define the bounding rectangle of the group. Only elements totally within the region are included.

Return Value

<int>

The number of elements in the group.

DM_ERROR if insufficient space remains in the form to create a group.

NULL if no elements can be found in the region.

Special Notes

The application may specify a temporary group of elements on which some operation may be performed. Only one group exists at a time, and will be destroyed if any elements operations are performed.

The coordinates of the group bounding box *must* be specified with x0,y0 as the top left hand corner of the region and x1,y1 as the lower right hand corner.

The bounding rectangle of the group will be set to the minimum region which encompasses its elements.

DeskMate Technical Reference Far Forms Resource

eform_delete_element (fpFORM_HDR, ElementTag)
FORM_HDR far *fpFORM_HDR;
int ElementTag;

eform_delete_element removes the element definition from the form.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.
ElementTag defines the element to be deleted.

Return Value

<int>
DM_ERROR if an invalid element tag is specified.

Special Notes

eform_delete_element removes the element definition from the form. If the element is an object, all elements of the object will be deleted. The tag for the deleted element will no longer be valid.

DeskMate Technical Reference Far Forms Resource

eform_delete_group (fpFORM_HDR)
FORM_HDR far *fpFORM_HDR;

eform_delete_group deletes all group elements from the form. Destroys the group element.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

Return Value

<int>

DM_ERROR if no group exists.

Special Notes

Operates exactly as **eform_delete_element** with the group treated as an object.

DeskMate Technical Reference Far Forms Resource

eform_delete_stroke (fpFORM_HDR, StrokeID)
FORM_HDR far *fpFORM_HDR;
int StrokeID;

eform_delete_stroke deletes the definition for the indicated stroke font character from the form's stroke definition list.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.
StrokeID is the identifier of the stroke character definition to be removed.

Return Value

<int>
DM_ERROR if an invalid **StrokeID** is specified.

Special Notes

The **StrokeID**'s are of the form ASCII character, font number. For instance, the character A of font three (3) would be identified by 4103 hex.

DeskMate Technical Reference

Far Forms Resource

eform_delete_unused_strokes (fpFORM_HDR)
FORM_HDR far *fpFORM_HDR;

eform_delete_unused_strokes deletes any of the form's stroke definitions which are not required by existing form text element definitions.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

Return Value

None.

Special Notes

eform_delete_unused_strokes examines all of the form's text elements to determine which stroke definitions are used. It then deletes the unused definitions to free space in the stroke list buffer.

DeskMate Technical Reference Far Forms Resource

eform_display_region (fpFORM_HDR, fpMAPRECT)
FORM_HDR far *fpFORM_HDR;
MAPRECT far *fpMAPRECT

eform_display_region displays any of the form's elements which intersect the specified region.

Input Parameters

fpFORM_HDR is a far pointer to the form to be displayed.

fpMAPRECT is a far pointer to a **MAPRECT** structure which holds the coordinates of the points that define the region to be displayed.

Return Value

None.

Special Notes

xorg holds x0, **xext** holds x1, **yorg** holds y0, **yext** holds y1.

eform_display_region does not require loading of the far forms resource.

DeskMate Technical Reference Far Forms Resource

eform_dup_element (fpFORM_HDR, ElementTag, fpFORM_DST)
FORM_HDR far *fpFORM_HDR;
int ElementTag;
FORM_DST far *fpFORM_DST;

eform_dup_element makes a scaled copy of the element in the specified bounding rectangle.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

ElementTag is the element tag.

fpFORM_DST is a far pointer to a structure which contains the (x0, y0) and (x1, y1) to define the bounding box to move the element into.

Return Value

<int>

The tag of the new element definition.

DM_ERROR is returned if insufficient space exist to copy the element or an invalid tag is specified.

Special Notes

eform_dup_element creates new element definitions, and inserts pointers in the form list. If the element is an object, all elements of the object will be affected.

Except for the special case of a line element, the destination coordinates must be specified with x0,y0 as the top left hand corner of the region and x1,y1 as the lower right hand corner.

DeskMate Technical Reference Far Forms Resource

eform_dup_group (fpFORM_HDR, fpFORM_DST)
FORM_HDR far *fpFORM_DST;
FORM_DST far *fpFORM_DST;

eform_dup_group creates a scaled copy of the group in the bounding box. Destroys the old group, replacing it with a new group containing the new elements.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

fpFORM_DST is a far pointer to the structure which contains the (x0, y0) and (x1, y1) which define the area to copy the group into.

Return Value

<int>

DM_ERROR if no group exists.

Special Notes

Operates exactly as **eform_copy_element** with the group treated as an object.

DeskMate Technical Reference Far Forms Resource

eform_element_to_bottom (fpFORM_HDR, ElementTag)
FORM_HDR far *fpFORM_HDR;
int ElementTag;

eform_element_to_bottom moves the element definition to the bottom of the element definitions so that the element will be drawn "under" the other elements.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.
ElementTag is the tag.

Return Value

<int>
DM_ERROR if an invalid element tag is specified.

Special Notes

eform_element_to_bottom shifts all element definitions up by the size of the moved element and then inserts the element definition under all other definitions. If the element is an object, all elements of the object will be affected. **eform_element_to_bottom** will alter all element offsets, but will not affect element tags.

DeskMate Technical Reference Far Forms Resource

eform_element_to_top (fpFORM_HDR, ElementTag)
FORM_HDR far *fpFORM_HDR;
int ElementTag;

eform_element_to_top moves the element definition to the top of the element definitions so that the element will be drawn "over" the other elements.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.
ElementTag is the element tag.

Return Value

<int>
DM_ERROR if an invalid element tag is specified.

Special Notes

eform_element_to_top performs the equivalent actions of **eform_copy_element** followed by deleting the original. If the element is an object, all elements of the object will be affected.

DeskMate Technical Reference

Far Forms Resource

eform_expand (fpFORM_HDR, HowMuchList, HowMuchStroke)
FORM_HDR far *fpFORM_HDR;
int HowMuchList, HowMuchStroke;

eform_expand expands the form display list to create free space between the form and element definitions, and adds additional room to the stroke list.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

HowMuchList is an integer indication of the number of additional bytes allocated for the display list.

HowMuchStroke is an integer indication of the number of additional bytes allocated for the stroke list.

Return Value

None.

Special Notes

eform_expand destroys the current group pointer.

DeskMate Technical Reference

Far Forms Resource

eform_find_element (fpFORM_HDR, x, y, n)
FORM_HDR far *fpFORM_HDR;
int x, y, n;

eform_find_element searches for elements (an object is also an element) which contain the point (x, y) within their bounding region. Returns a pointer to the nth satisfying element in the display list.

Input Parameters

fpFORM_HDR is a far pointer to the form to be searched.
n is which satisfying element to be returned.
x, y is the point from which to search.

Return Value

<int>
The tag of the nth satisfying element.
NULL if no element satisfies the search.

Special Notes

eform_find_element examines the bounding boxes of the element definitions. If the element is an object, only the bounding box of the object will be considered.

DeskMate Technical Reference Far Forms Resource

eform_find_stroke (fpFORM_HDR, StrokeID)
FORM_HDR far *fpFORM_HDR;
int StrokeID;

eform_find_stroke returns a pointer to the stroke character definition stored in the form stroke list.

Input Parameters

fpFORM_HDR is a far pointer to the form header structure.
StrokeID is the stroke definition identifier.

Return Value

A far pointer to the corresponding stroke definition.
DM_ERROR if the **StrokeID** can't be matched.

Special Notes

Generally the **StrokeID** should consist of a one byte ASCII value and the font number.

DeskMate Technical Reference Far Forms Resource

eform_get_extents (fpFORM_HDR)
FORM_HDR far *fpFORM_HDR;

eform_get_extents examines the bounding rectangles of all of the elements in a form, and returns the maximum x and y values.

Input Parameters

fpFORM_HDR is a far pointer to the form to be examined.

Return Value

<long integer>

The most significant word is the x extent. The least significant word is the y extent.

Special Notes

The form should be opened before calling **eform_get_extents**.
eform_get_extents does not require loading of the far forms resource.

DeskMate Technical Reference Far Forms Resource

eform_get_pointer (fpFORM_HDR, ElementTag)
FORM_HDR far *fpFORM_HDR;
int ElementTag;

eform_get_pointer returns a pointer to the beginning of the element definition.

Input Parameters

fpFORM_HDR is a far pointer to the form header structure of the form to be modified.
ElementTag is the tag of the element to be located.

Return Value

A far pointer to the element definition. The first byte of the definition indicates the definition type.
DM_ERROR will be returned if an invalid tag is specified.

Special Notes

The element definitions are stored exactly as the structures used to specify them with the following exception. Variable length definitions such as text and polyline do not hold a pointer to their elements, instead their elements are stored beginning contiguously with the definition, beginning at the location which would have been the element array pointer.

DeskMate Technical Reference Far Forms Resource

`eform_group_to_bottom` (`fpFORM_HDR`)
`FORM_HDR` far `*fpFORM_HDR`;

`eform_group_to_bottom` moves the element definitions of the group to the bottom of the element definitions so that the elements will be drawn "under" the other elements.

Input Parameters

`fpFORM_HDR` is a far pointer to the form to be modified.

Return Value

<int>

DM_ERROR if no group exists.

Special Notes

`eform_group_to_bottom` shifts all element definitions up by the size of the moved elements and then inserts the group element definitions under all other definitions. If the any element is an object, all elements of the object will be affected. **`eform_group_to_bottom`** will alter all element pointers.

DeskMate Technical Reference Far Forms Resource

form_group_to_top (p_form)
FORM_HDR *p_form;

form_group_to_top moves the element definitions of the group to the top of the element definitions so that the elements will be drawn "over" the other elements.

Input Parameters

tpFORM_HDR is a far pointer to the form to be modified.

Return Value

<int>

DM_ERROR if no group exists.

Special Notes

eform_group_to_top performs the equivalent actions of a **eform_copy_group** followed by deleting the original. If the element is an object, all elements of the object will be affected. **eform_group_to_top** will alter all element pointers above the original group element definitions.

DeskMate Technical Reference Far Forms Resource

eform_make_object (fpFORM_HDR)
FORM_HDR far *fpFORM_HDR;

eform_make_object creates an object from all of the elements in the group. Removes all tags to the individual elements from the form, replacing them with a single pointer to the object element. Destroys the group element.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

Return Value

<int>

The tag for the new element definition.

DM_ERROR is returned if no valid group exists.

Special Notes

Converts the group definition to an object definition. Inserts a pointer to the new object into the form. Deletes any references to the elements of the new object.

DeskMate Technical Reference Far Forms Resource

eform_mod_attr (fpFORM_HDR, ElementTag, fpFORM_ATTR)
FORM_HDR far *fpFORM_HDR;
int ElementTag;
FORM_ATTR far *fpFORM_ATTR;

eform_mod_attr modifies the specified attributes of the element.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

ElementTag is an element tag.

fpFORM_ATTR is a far pointer to the structure which contains the new attributes. Any attributes which have the value **DM_ERROR** will be unchanged.

Return Value

<int>

DM_ERROR if the element definition structure type does not match the element type.

Special Notes

eform_mod_attr modifies the attributes of the elements definitions, but does not have to change the pointers in the form list. If the element is an object, all elements of the object which are of the element definition type will be affected.

DeskMate Technical Reference

Far Forms Resource

eform_mod_grp_attr (fpFORM_HDR, fpFORM_Element)
FORM_HDR far *fpFORM_HDR;
FORM_Element far *fpFORM_Element;

eform_mod_grp_attr modifies the specified attributes of the element within the group.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

fpFORM_Element is a far pointer to the element definition structure of type **FORM_BEV**, **FORM_ELLIPSE**, **FORM_LINE**, **FORM_RECT**, **FORM_TEXT**, **FORM_U_ARC**, **FORM_U_IMAGE**, or **FORM_U_POLY**, which contains the new attributes. Any attributes of the element definition which contain **DM_ERROR** will be unchanged.

Return Value

<int>

DM_ERROR if the element definition structure type does not match any of the element types in the group.

Special Notes

eform_mod_grp_attr modifies the attributes of the group element definitions which are of the same type as the element definition. It does not have to change the pointers in the form list.

See Also

FORM_BEV structure.
FORM_ELLIPSE structure.
FORM_LINE structure.
FORM_RECT structure.
FORM_TEXT structure.
FORM_U_ARC structure.
FORM_U_IMAGE structure.
FORM_U_POLY structure.

DeskMate Technical Reference

Far Forms Resource

eform_move_element (fpFORM_HDR, ElementTag, fpFORM_DST)
FORM_HDR far *fpFORM_HDR;
int ElementTag;
FORM_DST far *fpFORM_DST;

eform_move_element moves and re-sizes the element into the new bounding rectangle. Updates the coordinate values of the element.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

ElementTag is the element tag.

fpFORM_DST is a far pointer to the destination structure which contains the coordinates (x0, y0) and (x1,y1) to define the bounding box to move the element into.

Return Value

<int>

DM_ERROR if an invalid tag is specified.

Special Notes

eform_move_element modifies the coordinates of the elements definitions, but does not have to change the pointers in the form list. If the element is an object, all elements of the object will be affected.

Except for the special case of a line element, the destination coordinates must be specified with x0,y0 as the top left hand corner of the rectangle and x1, y1 as the lower right corner.

DeskMate Technical Reference Far Forms Resource

eform_move_group (fpFORM_HDR, fpFORM_DST)
FORM_HDR far *fpFORM_HDR;
FORM_DST far *fpFORM_DST;

eform_move_group moves and re-sizes the group into the new bounding box. Updates the coordinate values of all of the group elements.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

fpFORM_DST is a far pointer to the structure which holds (x0, y0) and (x1, y1) which define the area to move the group into.

Return Value

<int>

DM_ERROR if no group exists.

Special Notes

Operates exactly as **eform_move_element** with the group treated as an object.

x0,y0 must be the top left hand corner of the destination region, x1,y1 must be the lower right hand region.

DeskMate Technical Reference

Far Forms Resource

eform_open (fpFORM_HDR)
FORM_HDR *pform;

eform_open initializes the indicated form buffer. It sets initial pointers to the element definition and the stroke character definitions.

Input Parameters

fpFORM_HDR is a far pointer to the header of the buffer for the list.

Return Value

None.

Special Notes

Prior to issuing **eform_open**, the caller must create a form header structure for the form. In this structure the following must be set by the user:

FORM_HDR.bNewList:

If this is set to zero, the forms resource will clear the display list buffer. If a previously created list is in the buffer, this should be set to 1 so the forms resource will calculate its internal pointers.

FORM_HDR.bVideo:

If this is 1 the forms resource will update the screen, if zero, no video output will take place.

FORM_HDR.list_size:

The size of the display list buffer in words.

FORM_HDR.stroke_size:

The size of the stroke list.

The remainder of the FORM_HDR elements will either be calculated by the forms resource or set by subsequent calls. These elements may be read, but should not be written to by the caller.

When the form is opened, the memory allocated may already contain a pre-created form display list. If so, the routine must determine the bottom of the form list and the top of the element definitions. If no display list is present, the buffer is initialized to all zeros.

WARNING: DO NOT MODIFY THE FOLLOWING FORM_HDR STRUCTURE ELEMENTS:

- next_entry
- last_def
- pGroup
- pFormList
- pStrokes

DeskMate Technical Reference Far Forms Resource

eform_paste (fpFORM_HDR)
FORM_HDR far *fpFORM_HDR;

eform_paste adds the clipboard display list to the form.

Input Parameters

fpFORM_HDR is a far pointer to the form to be modified.

Return Value

<int>

The tag of the object which contains the elements from the clipboard.
six if non-form data is in the clipboard, the tag will not be incremented.
DM_ERROR if insufficient space.

Special Notes

The elements pasted from the clipboard will be contained in an object.

DeskMate Technical Reference Far Forms Resource

eform_region_empty (fpFORM_HDR, fpMAPRECT)
FORM_HDR far *fpFORM_HDR;
MAPRECT far *fpMAPRECT;

eform_region_empty determines if the form contains any elements which intersect the indicated region.

Input Parameters

fpFORM_HDR is a far pointer to the form header structure of the form to be modified.

fpMAPRECT is a far pointer to a MAPRECT structure which contains the coordinates of the area to be searched.

Return Value

NULL if the region is empty.

One (1) if the form intersects the region specified by **pMAPRECT**.

DeskMate Technical Reference

Far Forms Resource

eform_replace_element (fpFORM_HDR, ElementTag, fpFORM_Element)
FORM_HDR far *fpFORM_HDR;
int ElementTag;
FORM_Element far *fpFORM_Element;

eform_replace_element replaces the definition of the element specified by **ElementTag** with the definition pointed to by **fpFORM_Element**. The new definition will be rendered in the same order as the replaced element.

Input Parameters

fpFORM_HDR is a far pointer to the form header structure of the form to be modified.

ElementTag is the tag of the element to be replaced.

fpFORM_Element is a far pointer to an element definition structure of type **FORM_BEV**, **FORM_ELLIPSE**, **FORM_LINE**, **FORM_RECT**, **FORM_TEXT**, **FORM_U_ARC**, **FORM_U_IMAGE**, or **FORM_U_POLY**.

Return Value

DM_ERROR if the **ElementTag** specified is not implemented or insufficient buffer space remains to store the new element definition.

Special Notes

eform_replace_element is used to preserve rendering order. A typical use is when replacing one text string with another.

DeskMate Technical Reference

Far Forms Resource

eform_scroll_down (fpFORM_HDR, WorldCordY)
FORM_HDR far *fpFORM_HDR;
int WorldCordY;

eform_scroll_down scrolls the indicated form down by the indicated number of world coordinates. The form will scroll within the window.

Input Parameters

fpFORM_HDR is a far pointer to the header structure of the desired form.
WorldCordY is the number of world coordinates to scroll the form.

Return Value

None.

Special Notes

eform_scroll_down activates the form's window, scrolls the information within that window, and then redraws the new data into the window. It then reactivates the caller's window.
eform_scroll_down does not require loading of the far forms resource.

DeskMate Technical Reference Far Forms Resource

eform_scroll_left (fpFORM_HDR, WorldCordX)
FORM_HDR far *fpFORM_HDR;
int WorldCordX;

eform_scroll_left scrolls the indicated form to the left by the indicated number of world coordinates. The form will scroll within the window.

Input Parameters

fpFORM_HDR is a far pointer to the header structure of the desired form.
WorldCordX is the number of world coordinates to scroll the form.

Return Value

None.

Special Notes

eform_scroll_left activates the form's window, scrolls the information within that window, and then redraws the new data into the window. It then reactivates the caller's window.
eform_scroll_left does not require loading of the far forms resource.

DeskMate Technical Reference

Far Forms Resource

eform_scroll_right (fpFORM_HDR, WorldCordX)
FORM_HDR far *fpFORM_HDR;
int WorldCordX;

eform_scroll_right scrolls the indicated form to the right by the indicated number of world coordinates. The form will scroll within the window.

Input Parameters

fpFORM_HDR is a far pointer to the header structure of the desired form.
WorldCordX is the number of world coordinates to scroll the form.

Return Value

None.

Special Notes

eform_scroll_right activates the form's window, scrolls the information within that window, and then redraws the new data into the window. It then reactivates the caller's window.
eform_scroll_right does not require loading of the far forms resource.

DeskMate Technical Reference Far Forms Resource

eform_scroll_up (fpFORM_HDR, WorldCordY)
FORM_HDR far *fpFORM_HDR;
int WorldCordY;

eform_scroll_up scrolls the indicated form up by the indicated number of world coordinates. The form will scroll within the window.

Input Parameters

fpFORM_HDR is a far pointer to the header structure of the desired form.
WorldCordY is the number of world coordinates to scroll the form.

Return Value

None.

Special Notes

eform_scroll_up activates the form's window, scrolls the information within that window, and then redraws the new data into the window. It then reactivates the caller's window.
eform_scroll_up does not require loading of the far forms resource.

DeskMate Technical Reference Far Forms Resource

```
eform_specify_group ( fpFORM_HDR, fpTagList, nTags )  
FORM_HDR far *fpFORM_HDR;  
int far *fpTagList;  
int nTags;
```

eform_specify_group makes the list of element tags into a group.

Input Parameters

fpFORM_HDR is a far pointer to the header structure of the desired form.

fpTagList is a far pointer to an integer tag list.

nTags is the number of tags pointed to by **fpTagList**.

Return Value

DM_ERROR if the group would not fit.

DeskMate Technical Reference Far Forms Resource

eform_update (fpFORM_HDR, ClearScreen)
FORM_HDR far *fpFORM_HDR;
int ClearScreen

eform_update draws (or redraws) the current contents of the form into the currently defined form window.

Input Parameters

fpFORM_HDR is a far pointer to the form to be displayed.

ClearScreen indicates whether or not the window will be cleared prior to update. If ClearScreen is ENABLED the screen will be cleared; if DISABLED the screen will not be cleared.

Return Value

None.

Special Notes

eform_scroll_left does not require loading of the far forms resource.

**Intelligent
Help Manager**

"Help Manager"

Table of Contents

General Description/Notes	13- 1
Defines	13- 2
help_disable_high ..Disables high and low level help.....	13- 3
help_enable_high ...Enables high and low level help.....	13- 4
hlp_notifyRegisters application notify routines.....	13- 5
hlp_stop_notifyTurns off the low level help.....	13- 7
ihm_add_stateAdds an application help state.....	13- 8
ihm_delete_state ...Deletes an application help state.....	13- 9
ihm_new_entryAdds a new entry to the help queue.....	13-10
ihm_statusReturns the type of component user is in....	13-12

DeskMate Technical Reference Intelligent Help Manager

General Description/Notes

The Intelligent Help Manager (IHM) enables the application to have context sensitive help within a DeskMate application. See the DeskMate Development Guide Help Section for a more complete description of the functionality.

All of the calls in this section are DeskMate 03.03.0x ONLY.

All of these help calls are linked in with the application from the DeskMate libraries, DM.LIB and DMMED.LIB. There are no binding calls the application needs to make in order to use these calls.

**DeskMate Technical Reference
Intelligent Help Manager**

DMGUF.H

Structures/Defines

```
/*-----*/
/*
/* CSRCMPS.H contains the Help structures and defines */
/*
/*-----*/

CMP_PUSHBUTTON      CMP_TAG+0
CMP_RADIOBUTTONS    CMP_TAG+1
CMP_ICONBUTTON      CMP_TAG+2
CMP_LISTBOX         CMP_TAG+3
CMP_CHECKBOX        CMP_TAG+4
CMP_EDIT            CMP_TAG+5
CMP_MENUBAR         CMP_TAG+6
CMP_MESSAGE         CMP_TAG+7

CMP_QUEUE_HELP      CMP_TAG+251
CMP_MESSAGE_BOX     CMP_TAG+252
CMP_DLG_BOX         CMP_TAG+253
CMP_MMBAR           CMP_TAG+254

/* structure for help notification */
struct notifyx
{
    int (far *pre) ();
    int (far *post) ();
    unsigned data seg;
    char *stack_ptr;
    unsigned psp;
}
typedef struct notifyx NOTIFY;
```

DeskMate Technical Reference Intelligent Help Manager

help_disable_high ()

help_disable_high disables ALL help (both high-level and low-level). High-level help is run by the application when it receives an **EVENT_APPL** telling it to run the help accessory. Low-level help is the help run by the CSR when the user presses F1 in a dialog box or message box.

Input Parameters

None.

Return Value

None.

Special Notes

The default state for help is ON. You MUST turn help back on with the **help_enable_high** call before exiting your application.

Examples

```
/* Make sure help is ON before I start */
help_enable_high();

/* turn help OFF in this dlg box */
help_disable_high();
ret = dlg_run(handle, pDlgBox);
/* now turn help back ON */
help_enable_high();
```

DeskMate Technical Reference

Intelligent Help Manager

help_enable_high ()

help_enable_high enables ALL help (both high-level and low-level). High-level help is run by the application when it receives an **EVENT_APPL** telling it to run the help accessory. Low-level help is the help run by the CSR when the user presses F1 in a dialog box or message box.

Input Parameters

None.

Return Value

None.

Special Notes

help_enable_high requires the application to link with DeskMate 03.03 or later libraries (DM.LIB or DMMED.LIB)

The default state for help is ON. You can call **help_enable_high** to make sure help is on while your application is running.

Examples

```
/* Make sure help is ON before I start */
help_enable_high();

/* turn help OFF in this dlg box */
help_disable_high();
ret = dlg_run(handle, pDlgBox);
/* now turn help back ON */
help_enable_high();
```

DeskMate Technical Reference Intelligent Help Manager

hlp_notify (pNOTIFY)
NOTIFY *pNOTIFY;

hlp_notify calls an application sub-routine before low-level help is executed from the CSR. It also calls a application sub-routine when low-level help from the CSR has completed. Low-level help is when help is invoked without the applications knowledge. For example: dialog boxes, menu bar and message boxes.

Input Parameters

pNOTIFY is a pointer to a NOTIFY structure.

Return Value

NULL if successful.

DM_ERROR if an error occurred adding the CSR event interpreter. In this case the application should call **help_disable_high** to disable help during message boxes and dialog boxes.

Special Notes

hlp_notify should be called by data base applications that will run on floppy disks. This allows the application to check and prompt for the data base files before returning to the application after low-level help. Otherwise, destruction of data on a diskette could occur.

The pre and post routines designated in the NOTIFY structure can not contain stack probes (calls to **_chkstk**). These routines can not call other routines that contain stack probes. If stack probes are present, a runtime error (stack overflow) will occur.

Example

The following example verifies that the correct data file is present after exiting low-level help.

```
char stack_buffer[1000];
struct notifyx
{
    int (far *pre) ();
    int (far *post) ();
    unsigned data_seg;
    char *stack_ptr;
    unsigned psp;
}
typedef struct notifyx NOTIFY;

int far Check_For_File_In_Drive;
int HelpNotifyOnFtag; /*return value of hlp_notify() */

extern unsigned _psp;

NOTIFY help_notification =
{
    0L,
    Check_For_File_In_Drive,
    0,
    &stack_buffer[999],
    0
}
```

DeskMate Technical Reference Intelligent Help Manager

```

};

int far hlp_notify( NOTIFY * );
int far hlp_stop_notify( );
main()
{
    csr_init();

    /* get data segment for NOTIFY structure */
    help_notification.data_seg = get_ds();

    /* get program segment prefix for the NOTIFY structure */
    help_notification.psp = _psp;

    /* notify help of the routines to call before running low-level */
    /* help and after exiting background help */
    if ( hlp_notify(&help_notification) == 0 )
        HelpNotifyOnFlag = TRUE;
    else
        HelpNotifyOnFlag = FALSE;

    /* if hlp_notify() returns an error, disable background */
    /* help during dialog boxes and message boxes */
    if ( HelpNotifyOnFlag == FALSE )
        help_disable_high();
        .
        .
        .
    /* do your stuff */
        .
        .
        .

    /* call hlp stop_notify() only if hlp_notify() was successful */
    if ( HelpNotifyOnFlag == TRUE )
        hlp_stop_notify();
    /* If hlp_notify() was unsuccessful, we disabled background */
    /* help for dialog boxes and message boxes. We must now enable */
    /* background help. */
    else
        help_enable_high();

    csr_end();
}

/*****
/* Check For File In Drive - This routine is called by Help when
/* low-level help was run on a message box or dialog box.
*****/
int far Check_For_File_In_Drive()
{
    FileAccessMsgBox.pMessage = "Insert disk containing data file";
    /* set message box for only the OK button */
    FileAccessMsgBox.btn_combo = MSG_COMBO_OK;

    /* run message box until file is found */
    while (fil_access(CurrentFile) == DM_ERROR)
    {
        /* display error message box */
        msg_run(&FileAccessMsgBox);
    }
}

```


DeskMate Technical Reference Intelligent Help Manager

hlp_stop_notify ()

hlp_stop_notify turns off the calls to the application sub-routines before and after low-level help is executed from the CSR.

Input Parameters

None.

Return Value

None.

See Also

hlp_notify()

DeskMate Technical Reference

Intelligent Help Manager

ihm_add_state (StateNum, pBuffer)

int StateNum;

char *pBuffer;

ihm_add_state will cause the help manager to save an application's current state. The state information will be used by the help accessory. An application state is assumed to be valid until the application deletes it with **ihm_delete_state**.

Input Parameters

StateNum indicates the state to add (The application can add up to 3 states). Valid numbers are 1, 2, and 3.

pBuffer is a pointer to a 4 byte buffer. The buffer contains the characters used to represent the state.

Return Value

DM_ERROR if the state was not added.

Special Notes

An application **must** delete its help states before exiting because the help manager will not clear the states when a new application starts. Therefore, your states could conflict with those from another application. Also, when your application is rerun, the old states may still exist.

ihm_add_state is useful when you want to add help information independent of the help queue. Calling **ihm_add_state** will overwrite any previous value for the state.

Examples

```
/* tell help user has empty file */
char sEmptyFile[5] = "UHEF"; /* stands for: user has empty file */
char sNonEmptyFile[5] = "NEF"; /* stands for: no empty file */

/* my app just started, set my first help state */
ihm_add_state(1, sEmptyFile);

/* the user added a record, update my help state */
ihm_add_state(1, sNonEmptyFile);
```

DeskMate Technical Reference

Intelligent Help Manager

ihm_delete_state (StateNum)

int StateNum;

ihm_delete_state causes the help manager to delete the application's state added by **ihm_add_state**. The state information will no longer be used by the help accessory. An application state is assumed to be valid until the application deletes it with this call.

Input Parameters

StateNum indicates the state to delete (The application can delete any of its 3 states). Valid numbers are 1, 2, and 3.

Return Value

DM_ERROR if the state was not deleted.

Special Notes

An application **must** delete its help states before exiting because the help manager will not clear the states when a new application starts. Therefore, your states could conflict with those from another application. Also, when your application is rerun, the old states may still exist.

Examples

```
/* about to exit the application, so delete */  
/* the help states created by this application */  
    ihm_delete_state(1);  
    ihm_delete_state(2);
```

DeskMate Technical Reference Intelligent Help Manager

ihm_new_entry (Type, pCMP)

int Type;

char *pCMP;

ihm_new_entry will add an entry into the help queue. The user of the call determines the entry type.

Input Parameters

Type is any of the valid component types, CMP_CHECKBOX, CMP_DLG_BOX, CMP_EDIT, CMP_ICONBUTTON, CMP_LISTBOX, CMP_MENUBAR, CMP_MESSAGE_BOX, CMP_MMBAR, CMP_PUSHBUTTON, CMP_RADIOBUTTONS. **Type** can also be a special entry CMP_QUEUE_HELP that indicates a help queue state that is independent of components.

pCMP_HEADER is a pointer to the corresponding component structure that goes with the specified component **Type**. If CMP_QUEUE_HELP **Type** is used the IHM expects **pCMP_HEADER** to be the pointer to a string. Only 20 bytes of the string will be copied into the queue for the entry.

Return Value

None.

Special Notes

If your application has an event interpreter or is a resource, you may need to guarantee the integrity of the queue. To guarantee the integrity of the queue, make sure the help manager is NOT IN THE MIDDLE OF AN ENTRY when **ihm_new_entry** is made. To do this use **ihm_status**. The return value from **ihm_status** should be zero.

ihm_new_entry is useful when you want to add help information that the Help Manager does not know about. You should only use this call if you intend to use the information you add when you write a help file for your application. For example, the Draw application uses this call to indicate when a tool is selected AND when the user is on the canvas. With this information, Draw help can then give tool help when a tool is selected, and more general help when the user has moved to the canvas.

Remember, the information added is eventually lost as the Help Manager adds more entries to its queue.

DeskMate Technical Reference Intelligent Help Manager

Examples

```
if (ihm_status() == 0) /* help manager is not in the middle of an
                        entry, so add a listbox entry */
    ihm_new_entry(CMP_LISTBOX, pMyListBox);

if (ihm_status() == 0) /* add a dialog box entry */
    ihm_new_entry(CMP_DLG_BOX, pMyDlgBox);

if (ihm_status() == 0)
{
    /* if help manager is not in a current entry, add
       these special entries to indicate the current state */
    ihm_new_entry(CMP_QUEUE_HELP, "text selected");
    ihm_new_entry(CMP_QUEUE_HELP, "on first object");
}
```

DeskMate Technical Reference

Intelligent Help Manager

ihm_status ()

ihm_status returns the type of the component the user is currently in, at the time the call is made.

Input Parameters

None.

Return Value

Zero if not in a component, otherwise the component tags below are returned.

CMP_PUSHBUTTON	CMP_TAG+0	
CMP_RADIOBUTTONS	CMP_TAG+1	
CMP_ICONBUTTON	CMP_TAG+2	
CMP_LISTBOX	CMP_TAG+3	
CMP_CHECKBOX	CMP_TAG+4	
CMP_EDIT	CMP_TAG+5	
CMP_MENUBAR	CMP_TAG+6	(in a component menu bar)
CMP_MESSAGE	CMP_TAG+7	
CMP_QUEUE_HELP	CMP_TAG+251	
CMP_MESSAGE_BOX	CMP_TAG+252	
CMP_DLG_BOX	CMP_TAG+253	
CMP_MMBAR	CMP_TAG+254	(in the main menu bar)

Example

```
WhereAmI = ihm_status();
```

Information Box Manager

"Information Box Manager"

Table of Contents

General Description/Notes	14- 1
Structures/Defines	14- 2
info_draw	Displays the information box on the screen...14- 5
info_draw_icon ...	Displays a draw list icon.....14- 6
info_put_string ..	Displays a word wrapped string.....14- 7

DeskMate Technical Reference Information Box Manager

General Description/Notes

These routines support the displaying of non-interactive messages on the screen. The information boxes support text strings, bitmaps, stroke fonts, and static boxes. The information box is drawn relative to the entire screen. Information boxes are very useful in demos and tutorials.

The text strings support embedded attributes such as **BOLD**, **UNDERLINE**, and **INVERSE**. These attributes are in the character range of C0h through C8h. The "normal" attribute will cause all other previous attributes to return to the "off" state. All other attribute characters will have a toggling effect within the string.

DeskMate Technical Reference Information Box Manager

Structures/Defines

```
/*-----*/  
/*  
/* CSRCMPS.H contains the Information box structures and defines */  
/*  
/*-----*/
```

The elements of an information string structure designate the location and color of the string, and identify the pointer to the string content.

Static boxes visually separate similar or related component groups within dialog boxes. For example, a radio button group is usually surrounded by a static box to separate it from other components in a dialog box.

```
/* Static box */  
struct static_box_defn  
{  
    char    type;        /* type of box */  
    MAPRECT maprect;     /* org/ext of box */  
    char    color;       /* color of the box */  
};  
typedef struct static_box_defn STATIC_BOX;
```

type:

Describes the kind of box to be used. The following identify the acceptable values for the type element in the STATIC_BOX structure:

```
/* Box types */  
FLAT_BOX      0        /* single thin line box */  
RAISED_BOX    1        /* 3-D raised block */  
PYRAMID       2        /* 3-D pyramid */  
DEST_FLAT_BOX 3        /* thin line box that erases a RAISED_BOX */  
LISTBOX_BOX   4        /* item list border for list boxes */  
GRAYED_BOX    0x80     /* flag to make boxes grayed */  
ENLARGED_BOX  0x40     /* flag to pad frame interior with gap */
```

MAPRECT:

Defines the location and size of the box. The MAPRECT structure is defined in CSRBASE.H and should be included in the application.

color:

Selects a valid color (COLOR1 to COLOR16) for the interior of the box.

```
struct info_string_defn  
{  
    MAPRECT    maprect;    /* word wrap boundaries */  
    char        color;     /* foreground color of string */  
    unsigned char *pString; /* pointer to zero terminated string */  
};  
typedef struct info_string_defn INFO_STRING;
```

MAPRECT:

Defines the location and size of the information box.

color:

Defines the foreground color (COLOR1 to COLOR16) of the string in the information box.

pString:

The pointer to the null-terminated string that will be displayed in an information box.

DeskMate Technical Reference Information Box Manager

The attribute controlled by most of these values is apparent from the defined name; for instance, INFO_NORMAL sets the normal attribute for the information box, INFO_BOLD sets the bold attribute for the information box.

```
/* Information Box String attribute defines */
INFO_NORMAL      0xC0      /* info box NORMAL attribute */
INFO_BOLD        0xC1      /* info box BOLD attribute */
INFO_UNDERLINE    0xC3      /* info box UNDERLINE attribute */
INFO_INVERSE      0xC4      /* info box INVERSE attribute */
INFO_GRAYED       0xC5      /* info box GRAYED attribute */
INFO_TRANSPARENT  0xC6      /* info box TRANSPARENT attribute */
INFO_RES1         0xC7      /* info box RESERVED 1 attribute */
INFO_RES2         0xC8      /* info box RESERVED 2 attribute */
```

```
struct info_icon_defn
{
    STATIC_ICON icon;          /* icon definition structure */
    char line_type;           /* line type for stroke list */
    char line_width;          /* line width for stroke list */
    char line_color;          /* line color for stroke list */
};
typedef struct info_icon_defn INFO_ICON;
```

icon:
The static icon definition and location.

line_type:
The line type to be used for the stroke list.

line_width:
Specifies the line width to be used for the stroke list.

line_color:
Defines the line color to be used for the stroke list.

```
struct image_defn
{
    MAPRECT maprect;          /* maprect on device to size image to */
    char bTransparent;         /* flag for COLOR1 transparency */
    int bitmap_xext;           /* number of pels in x extent of image */
    int bitmap_yext;           /* number of pels in y extent of image */
    char nColors;              /* IMAGE 2COLORS, IMAGE 4COLORS, etc. */
    char *pBitmap;             /* pointer to generic bitmap */
};
typedef struct image_defn IMAGE;
```

```
/* Number of colors in a bitmap definition */
IMAGE_2COLORS    1
IMAGE_4COLORS    2
IMAGE_16COLORS   4
IMAGE_256COLORS  8
```

An information box is a static box that conveys a specific message to the user. The application controls how long to leave the message on the screen and is responsible for saving and restoring the background.

```
struct info_box_defn
{
    STATIC_BOX frame;          /* STATIC BOX for INFO BOX frame */
    char bgnd_color;           /* background color of box */
    char nStrings;             /* number of strings in INFO BOX */
    INFO_STRING *pStrings;     /* pointer to INFO STRING struc(s) */
    char nBoxes;               /* number of static boxes in INFO_BOX */
};
```

DeskMate Technical Reference

Information Box Manager

```
STATIC_BOX *pBoxes;      /* pointer to STATIC_BOX struc(s) */
char        nIcons;      /* number of icon images in INFO_BOX */
INFO_ICON   *pIcons;     /* pointer to STATIC_ICON struc(s) */
char        nImages;     /* number of generic bitmaps in INFO_BOX */
IMAGE       *pImages;    /* pointer to IMAGE struc(s) */
};
typedef struct info_box_defn INFO_BOX;
```

INFO_BOX.STATIC_BOX.frame:

The frame placed around the information box is one of the "static boxes". This element is a `STATIC_BOX` structure. If no frame is desired around the information box then use the `INFO_NO_FRAME` define for the static box type. The `STATIC_BOX` maprect is relative to the overall frame's maprect.

`INFO_NO_FRAME` `CSR_ERROR` /* no frame option for info boxes */

INFO_BOX.bgnd_color:

Specifies the background color of the information box. This color is used to clear the interior of the information box and the background of the information box frame. `COLOR1-COLOR16`.

INFO_BOX.nStrings:

Specifies the number of word-wrapped strings within the information box.

INFO_BOX.INFO_STRING.pStrings:

A pointer to a number of consecutive `INFO_STRING` structures as specified by the `nStrings` element. The `INFO_STRING` maprect is relative to the overall frame's maprect.

INFO_BOX.nBoxes:

Specifies the number of static boxes within the information box.

INFO_BOX.STATIC_BOX.pBoxes:

A pointer to a number of consecutive `STATIC_BOX` structures as specified by the `nBoxes` element. (See above)

INFO_BOX.nIcons:

Specifies the number of icon images in `INFO_BOX`.

INFO_BOX.INFO_ICON.pIcons:

A pointer to a number of `INFO_ICON` structure(s) as specified by the `nIcons` element.

INFO_BOX.nImages:

Number of generic bitmaps within the `INFO_BOX`.

INFO_BOX.IMAGE.pImages:

A pointer to `IMAGE` structures(s).

DeskMate Technical Reference

Information Box Manager

info_draw (pINFO_BOX)
INFO_BOX *pINFO_BOX;

info_draw displays information for the user using word wrapped text with embedded attributes, static boxes, draw list images, and/or bitmap images.

Input Parameters

pINFO_BOX is a pointer to an **INFO_BOX** structure.

Return Value

None.

Special Notes

The information box displayed is non-interactive and control is passed immediately back to the caller. The caller is responsible for saving the screen and removing the information box from the screen.

See Also

INFO_BOX structure.

Bug

Information box strings will not display correctly if two consecutive new line sequences are contained within the string. To work around this, use a NEW LINE, SPACE, NEW LINE sequence (i.e. \r\n \r\n).

DeskMate Technical Reference Information Box Manager

```
info_draw_icon ( pINFO_ICON )  
INFO_ICON *pINFO_ICON;
```

info_draw_icon draws a draw list icon with independent line attributes.

Input Parameters

pINFO_ICON is a pointer to an **INFO_ICON** structure.

Return Value

None.

Special Notes

None of the attributes of the currently active window will be modified by **info_draw_icon**.

See Also

INFO_ICON structure.

DeskMate Technical Reference Information Box Manager

info_put_string (pINFO_STRING)
INFO_STRING *pINFO_STRING;

info_put_string displays a word-wrapped string with or without embedded attributes within a specified maprect.

Input Parameters

pINFO_STRING is a pointer to an **INFO_STRING** structure.

Return Value

None.

Special Notes

The string is placed at the origin of the window and word-wrapped at the right edge of the window (as specified by the **xext** element of the **MAPRECT** structure). If the end of the string does not fit within the **yext** of the **MAPRECT**, then that portion of the string will be clipped.

The background color of the string is assumed to be the background color of the currently active window.

See Also

INFO_STRING structure.

Keyboard Manager

"Keyboard Manager"

Table of Contents

General Description/Notes	15- 1
Defines	15- 3
kbd_flushClears the keyboard buffer.....	15- 8
kbd_readReads a single CSR extended key code.....	15- 9
kbd_scanScans the keyboard for an extended key code..	15-10

DeskMate Technical Reference

Keyboard Manager

General Description/Notes

The keyboard manager makes direct CSR keyboard access available to the application and performs internal keyboard functions for the event manager. All key codes returned by the keyboard manager are CSR extended key codes, as described under keyboard manager Defines.

The direct keyboard routines make the following conversions:

Keystroke	BIOS Code	CSR Extended Code
BREAK	0000h	BRK KEY
PG UP	4900h	PGUP KEY
PGUP KEY (as converted above)	-----	CTRL UP ARROW
PG DN	5100h	PGDN KEY
PGDN KEY (as converted above)	-----	CTRL DOWN ARROW
SHIFT-space bar	-----	SPACE BAR
SHIFT-BACK SPACE	-----	BKSPACE KEY

In addition to the above list, the direct keyboard routines convert other key codes according to the following rules:

1. ALT'ed a-z are converted to (ALT_TAG) + the ASCII code of the upper case character. For example, ALT-A (2E00h) is converted to 0FD41h (assuming ALT_TAG = 0FD00h), or ALT_TAG + the ASCII code for 'A'.
2. All remaining extended codes are converted by moving the extended code from the high order byte to the low order byte and adding SYS_TAG. For example, the F4 key code is converted to F4, or SYS_TAG + (F4 key code / 100h).
3. All non-extended codes with a character code below 1Ah and a scan code between 10h and 32h (inclusive) are converted by clearing the scan code and adding CTRL_TAG. For example, CTRL-C is converted CTRL_C, or CTRL_TAG + (CTRL-C key code MOD 100h).
4. All remaining key codes are converted by clearing the scan code and adding (ASCII_TAG). For example, Q is converted to 'Q', or ASCII_TAG + (Q key code MOD 100h). ASCII_TAG will always be 0000h.

Internal conversions made for the Event Manager only:

Keystroke	CSR Extended Code
SHIFT-PG UP	SHFT PAGE UP
SHIFT-PG DN	SHFT PAGE DOWN
SHIFT-HOME	SHFT HOME
SHIFT-END	SHFT END

DeskMate Technical Reference Keyboard Manager

In addition, the following key codes are converted when they immediately follow the F11 or SHIFT F1 key:

Keystroke Following F11	CSR Extended Code
v (lower case only)	F11 TAG +'v'
all others (low case only)	F11 TAG + char

Note, the Event Manager cannot return the F11 or SHIFT F1 keys as an event.

SHIFT_F1+v displays the DMCSR version number on the time area of the title line.

DeskMate Technical Reference Keyboard Manager

Defines

```
/*-----*/  
/*  
/* CSRKEYS.H contains the Key codes */  
/*  
/*-----*/
```

```
/* CTRL'ed A-Z */  
CTRL_A (CTRL_TAG+'A'-0x40)  
CTRL_B (CTRL_TAG+'B'-0x40)  
CTRL_C (CTRL_TAG+'C'-0x40)  
CTRL_D (CTRL_TAG+'D'-0x40)  
CTRL_E (CTRL_TAG+'E'-0x40)  
CTRL_F (CTRL_TAG+'F'-0x40)  
CTRL_G (CTRL_TAG+'G'-0x40)  
CTRL_H (CTRL_TAG+'H'-0x40)  
CTRL_I (CTRL_TAG+'I'-0x40)  
CTRL_J (CTRL_TAG+'J'-0x40)  
CTRL_K (CTRL_TAG+'K'-0x40)  
CTRL_L (CTRL_TAG+'L'-0x40)  
CTRL_M (CTRL_TAG+'M'-0x40)  
CTRL_N (CTRL_TAG+'N'-0x40)  
CTRL_O (CTRL_TAG+'O'-0x40)  
CTRL_P (CTRL_TAG+'P'-0x40)  
CTRL_Q (CTRL_TAG+'Q'-0x40)  
CTRL_R (CTRL_TAG+'R'-0x40)  
CTRL_S (CTRL_TAG+'S'-0x40)  
CTRL_T (CTRL_TAG+'T'-0x40)  
CTRL_U (CTRL_TAG+'U'-0x40)  
CTRL_V (CTRL_TAG+'V'-0x40)  
CTRL_W (CTRL_TAG+'W'-0x40)  
CTRL_X (CTRL_TAG+'X'-0x40)  
CTRL_Y (CTRL_TAG+'Y'-0x40)  
CTRL_Z (CTRL_TAG+'Z'-0x40)
```

CTRL_TAG = 0FDD00h

```
/* ALT'ed A-Z */  
ALT_A (ALT_TAG+'A')  
ALT_B (ALT_TAG+'B')  
ALT_C (ALT_TAG+'C')  
ALT_D (ALT_TAG+'D')  
ALT_E (ALT_TAG+'E')  
ALT_F (ALT_TAG+'F')  
ALT_G (ALT_TAG+'G')  
ALT_H (ALT_TAG+'H')  
ALT_I (ALT_TAG+'I')  
ALT_J (ALT_TAG+'J')  
ALT_K (ALT_TAG+'K')  
ALT_L (ALT_TAG+'L')  
ALT_M (ALT_TAG+'M')  
ALT_N (ALT_TAG+'N')  
ALT_O (ALT_TAG+'O')  
ALT_P (ALT_TAG+'P')  
ALT_Q (ALT_TAG+'Q')  
ALT_R (ALT_TAG+'R')  
ALT_S (ALT_TAG+'S')  
ALT_T (ALT_TAG+'T')  
ALT_U (ALT_TAG+'U')  
ALT_V (ALT_TAG+'V')  
ALT_W (ALT_TAG+'W')  
ALT_X (ALT_TAG+'X')  
ALT_Y (ALT_TAG+'Y')  
ALT_Z (ALT_TAG+'Z')
```

ALT_TAG = 0FE000h

DeskMate Technical Reference Keyboard Manager

```
/* Arrow keys */
UP_ARROW          (SYS_TAG+0x48)
DOWN_ARROW        (SYS_TAG+0x50)
LEFT_ARROW        (SYS_TAG+0x4B)
RIGHT_ARROW       (SYS_TAG+0x4D)
```

```
SHFT_UP_ARROW     (WDG_TAG+0x0F)
SHFT_DOWN_ARROW   (WDG_TAG+0x12)
SHFT_LEFT_ARROW   (WDG_TAG+0x10)
SHFT_RIGHT_ARROW  (WDG_TAG+0x11)
```

```
CTRL_UP_ARROW     OLD_PGUP_KEY
CTRL_DOWN_ARROW   OLD_PGDN_KEY
CTRL_LEFT_ARROW   (SYS_TAG+0x73)
CTRL_RIGHT_ARROW  (SYS_TAG+0x74)
```

```
ALT_UP_ARROW      (WDG_TAG+0x15)
ALT_DOWN_ARROW    (WDG_TAG+0x16)
ALT_LEFT_ARROW    (WDG_TAG+0x17)
ALT_RIGHT_ARROW   (WDG_TAG+0x18)
```

```
SHFT_CTRL_UP_ARROW (WDG_TAG+0x1B)
SHFT_CTRL_DOWN_ARROW (WDG_TAG+0x1C)
SHFT_CTRL_LEFT_ARROW (WDG_TAG+0x1D)
SHFT_CTRL_RIGHT_ARROW (WDG_TAG+0x1F)
```

/* Function keys */

```
SHFT_F1          (SYS_TAG+0x54)
SHFT_F2          (SYS_TAG+0x55)
SHFT_F3          (SYS_TAG+0x56)
SHFT_F4          (SYS_TAG+0x57)
SHFT_F5          (SYS_TAG+0x58)
SHFT_F6          (SYS_TAG+0x59)
SHFT_F7          (SYS_TAG+0x5A)
SHFT_F8          (SYS_TAG+0x5B)
SHFT_F9          (SYS_TAG+0x5C)
SHFT_F10         (SYS_TAG+0x5D)
```

```
F1              (SYS_TAG+0x3B)
F2              (SYS_TAG+0x3C)
F3              (SYS_TAG+0x3D)
F4              (SYS_TAG+0x3E)
F5              (SYS_TAG+0x3F)
F6              (SYS_TAG+0x40)
F7              (SYS_TAG+0x41)
F8              (SYS_TAG+0x42)
F9              (SYS_TAG+0x43)
F10             (SYS_TAG+0x44)
```

```
CTRL_F1         (SYS_TAG+0x5E)
CTRL_F2         (SYS_TAG+0x5F)
CTRL_F3         (SYS_TAG+0x60)
CTRL_F4         (SYS_TAG+0x61)
CTRL_F5         (SYS_TAG+0x62)
CTRL_F6         (SYS_TAG+0x63)
CTRL_F7         (SYS_TAG+0x64)
CTRL_F8         (SYS_TAG+0x65)
CTRL_F9         (SYS_TAG+0x66)
CTRL_F10        (SYS_TAG+0x67)
```

```
ALT_F1          (SYS_TAG+0x68)
ALT_F2          (SYS_TAG+0x69)
ALT_F3          (SYS_TAG+0x6A)
```

SYS_TAG = 0FF00h

DeskMate Technical Reference Keyboard Manager

```
ALT_F4      (SYS_TAG+0x6B)
ALT_F5      (SYS_TAG+0x6C)
ALT_F6      (SYS_TAG+0x6D)
ALT_F7      (SYS_TAG+0x6E)
ALT_F8      (SYS_TAG+0x6F)
ALT_F9      (SYS_TAG+0x70)
ALT_F10     (SYS_TAG+0x71)
```

```
/* HOME, END, PGUP and PGDN keys */
HOME_KEY    (SYS_TAG+0x47)
END_KEY      (SYS_TAG+0x4F)
PGUP_KEY     OLD_CTRL_UP_ARROW
PGDN_KEY     OLD_CTRL_DOWN_ARROW
```

```
SHFT_HOME_KEY (WDG_TAG+0x0E)
SHFT_END_KEY   (WDG_TAG+0x0D)
SHFT_PGUP_KEY  (WDG_TAG+0x0B)
SHFT_PGDN_KEY  (WDG_TAG+0x0C)
```

WDG_TAG = 8400h

```
CTRL_HOME_KEY (SYS_TAG+0x77)
CTRL_END_KEY   (SYS_TAG+0x75)
CTRL_PGUP_KEY  (WDG_TAG+0x00)
CTRL_PGDN_KEY  (SYS_TAG+0x76)
```

```
SHFT_CTRL_HOME_KEY (WDG_TAG+0x20)
SHFT_CTRL_END_KEY   (WDG_TAG+0x21)
SHFT_CTRL_PGUP_KEY  (WDG_TAG+0x22)
SHFT_CTRL_PGDN_KEY  (WDG_TAG+0x23)
```

```
/* INSERT and DELETE keys */
INSERT_KEY (SYS_TAG+0x52)
DELETE_KEY (SYS_TAG+0x53)
```

```
SHFT_INSERT_KEY (WDG_TAG+0x24)
SHFT_DELETE_KEY (WDG_TAG+0x25)
```

```
CTRL_INSERT_KEY (WDG_TAG+0x19)
CTRL_DELETE_KEY (WDG_TAG+0x1A)
```

```
/* SPACEBAR, BACKSPACE and TAB keys */
SPACE_KEY (ASCII_TAG+0x20)
BKSPACE_KEY (ASCII_TAG+0x08)
TAB_KEY     (ASCII_TAG+0x09)
```

ASCII_TAG = 0000h

```
SHFT_SPACE_KEY (WDG_TAG+0x04)
SHFT_BKSPACE_KEY (WDG_TAG+0x01)
SHFT_TAB_KEY    (SYS_TAG+0x0F)
```

```
CTRL_SPACE_KEY (WDG_TAG+0x05)
```

```
ALT_TAB_KEY (SYS_TAG+0x8E)
ALT_BKSPACE_KEY (WDG_TAG+0x26)
```

```
/* Miscellaneous keys */
SHFT_RETURN_KEY (WDG_TAG+0x02)
BRK_KEY          (SYS_TAG+0x80)
PRINT_KEY        (ASCII_TAG+0x10)
ABORT_KEY        CSR_ERROR
ALT_EQUAL        (SYS_TAG+0x83)
```

CSR_ERROR = -1

```
/* Extended key codes */
EC_PREVIOUS ALT_P
EC_NEXT      ALT_N
EC_FIRST     ALT_F
```

DeskMate Technical Reference Keyboard Manager

EC_LAST	ALT_L
EC_SAVE	ALT_S
EC_QUIT	ESC_KEY
EC_MOVE_FOCUS_FWRD	TAB_KEY
EC_MOVE_FOCUS_BKWD	SHFT_TAB_KEY
EC_SELECT	SPACE_KEY
EC_CANCEL	ESC_KEY
EC_TAB_CURSOR	TAB_KEY
EC_BACKSPACE	BKSPACE_KEY
EC_BACK_TAB	SHFT_TAB_KEY
EC_INSERT	INSERT_KEY
EC_DELETE	DELETE_KEY
EC_HOME_CURSOR	CTRL_HOME_KEY
EC_END_CURSOR	CTRL_END_KEY
EC_LINE_START	HOME_KEY
EC_LINE_END	END_KEY
EC_CURSOR_UP	UP_ARROW
EC_CURSOR_DOWN	DOWN_ARROW
EC_CURSOR_LEFT	LEFT_ARROW
EC_CURSOR_RIGHT	RIGHT_ARROW
EC_SELECT_UP	SHFT_UP_ARROW
EC_SELECT_DOWN	SHFT_DOWN_ARROW
EC_SELECT_LEFT	SHFT_LEFT_ARROW
EC_SELECT_RIGHT	SHFT_RIGHT_ARROW
EC_SELECT_HOME	SHFT_CTRL_HOME_KEY
EC_SELECT_END	SHFT_CTRL_END_KEY
EC_SEL_LINE_START	BIG_SLCT_HOME_KEY
EC_SEL_LINE_END	BIG_SLCT_END_KEY
EC_GROUP_SEL_UP	SHFT_CTRL_UP_ARROW
EC_GROUP_SEL_DOWN	SHFT_CTRL_DOWN_ARROW
EC_GROUP_SEL_LEFT	SHFT_CTRL_LEFT_ARROW
EC_GROUP_SEL_RIGHT	SHFT_CTRL_RIGHT_ARROW
EC_PAGE_SEL_UP	BIG_SLCT_UP_ARROW
EC_PAGE_SEL_DOWN	BIG_SLCT_DOWN_ARROW
EC_PAGE_SEL_LEFT	SHFT_CTRL_PGUP_KEY
EC_PAGE_SEL_RIGHT	SHFT_CTRL_PGDN_KEY
EC_PAGE_UP	PGUP_KEY
EC_PAGE_DOWN	PGDN_KEY
EC_PAGE_LEFT	CTRL_PGUP_KEY
EC_PAGE_RIGHT	CTRL_PGDN_KEY
EC_CUT	SHFT_DELETE_KEY
EC_PASTE	SHFT_INSERT_KEY
EC_COPY	CTRL_INSERT_KEY
EC_CLEAR	DELETE_KEY
EC_ALT_UP	ALT_UP_ARROW
EC_ALT_DOWN	ALT_DOWN_ARROW
EC_ALT_LEFT	ALT_LEFT_ARROW
EC_ALT_RIGHT	ALT_RIGHT_ARROW
EC_GO	RETURN_KEY

DeskMate Technical Reference Keyboard Manager

EC GROUP UP
EC GROUP DOWN
EC GROUP LEFT
EC GROUP RIGHT

CTRL UP ARROW
CTRL DOWN ARROW
CTRL LEFT ARROW
CTRL RIGHT ARROW

DeskMate Technical Reference

Keyboard Manager

kbd_flush ()

kbd_flush clears the keyboard buffer.

Input Parameters

None.

Return Value

None.

Special Notes

kbd_flush clears the keyboard buffer only, it has no effect on any events that may be buffered in the event manager queue.

DeskMate Technical Reference Keyboard Manager

kbd_read ()

kbd_read reads a single CSR extended key code.

Input Parameters

None.

Return Value

<int>

An extended (EC_) key code.

Special Notes

The character read is removed from the keyboard buffer.

Example

```
int key;  
  
/* read a key from the keyboard */  
key = kbd_read();
```

DeskMate Technical Reference Keyboard Manager

kbd_scan ()

kbd_scan scans the keyboard and immediately returns a CSR extended key code if available.

Input Parameters

None.

Return Value

<int>

An extended key (EC_) code if available.

CSR_ERROR if no key code available.

Special Notes

The character read is **NOT** removed from the keyboard buffer.

Example

```
int key;  
  
/* scan the keyboard for a key */  
key = kbd_scan();
```

Library Functions

"Library Functions"

Table of Contents

General Description/Notes	16- 1
Structures/Defines	16- 2
about_versions	Displays the About dialog box.....16- 4
cnvrt_to_julian	Converts standard to julian format.....16- 5
cnvrt_to_std	Converts julian to standard format.....16- 6
compare_chars	Compares specified characters.....16- 7
compare_chars_ins ..	Compares case insensitive characters.....16- 8
dmmore_add_accessory .	Adds an accessory to the More accessory..16- 9
dmmore_delete_accessory .	Deletes an acc from the More acc.....16-11
get_ds	Gets the current data segment value...16-12
get_intl	Gets international structure information...16-13
get_month_strings ..	Gets the month abbreviations.....16-14
PageSetupDialog	Display DeskMate's Page Setup dialog box...16-15
str_upper	Converts a string to upper case.....16-16
to_upper	Converts a character to upper case.....16-17
waitloop	Waits for a specified period of time.....16-18

General Description/Notes

The functions contained in this section are included in the DeskMate libraries, DM.LIB and DMMED.LIB.

Date Conversion

These routines allow an application to convert from standard mm/dd/yyyy date format to julian date format. These routines are:

```
cnvrt_to_julian()
cnvrt_to_std()
get_month_strings()
```

The intended use of `cnvrt_to_julian` and `cnvrt_to_std` is to provide a uniform date field input to the database, without putting the conversion and validation code into the database manager.

If a date is in julian format and `cnvrt_to_std` is called, the international structure may be used to format the date according to the current country settings.

International Support

The international support routines provide support for the international versions of DeskMate. These routines are:

```
compare_chars()
compare_chars_ins()
get_intl()
str_upper()
to_upper()
```

These routines are part of the GUF resource, and therefore require the application to call `guf_bind_init` before any of the following calls may be made.

There are three general areas which affect code development when working in an international environment. These are:

1. Uniform uppercase translations.
2. Collating sequence character comparison.
3. DOS standard international information such as date and time format.

The above routines have been provided to serve these areas of code development.

More Accessory

The file, DMMORE.CFG, will be used to display a list box of accessories that vendors desire to be available for use with their application. The vendor will make the following calls to add an entry to the More Accessories list box or delete an entry from the More Accessories list box:

```
dmmore_add_accessory
dmmore_delete_accessory
dmmore_get_label
dmmore_get_basename
```

DeskMate Technical Reference

Library Functions

dmmore_get_basename (pLabel, pBasename)

char *pLabel;

char *pBasename;

dmmore_get_basename returns the basename associated with a label after the error **DMMORE_LABEL_NOT_UNIQUE** is returned from **dmmore_add_accessory**.

Input Parameters

pLabel is a pointer to the label to be searched for so that its associated basename may be found.

pBasename is a pointer a buffer to which the desired basename will be copied. The minimum buffer size is 9 bytes.

Return Value

<int>

DMMORE_OK if the basename has been copied to the buffer pointed to by **pBasename**.

DMMORE_FILE_NOT_IN_MEMORY if the **DMMORE.CFG** environment has not been loaded into memory. This call should only be made after an error, **DMMORE_LABEL_NOT_UNIQUE**, has been returned from **dmmore_add_accessory**.

DMMORE_LABEL_NOT_FOUND if the label pointed to by **pLabel** does not match any labels in the **dmmore** environment.

Special Notes

dmmore_get_basename requires the application to link with DeskMate 03.05 or later libraries (**DM.LIB** or **DMMED.LIB**).

dmmore_get_basename checks to see if the **dmmore** environment is currently in memory and if it is searches the **dmmore** environment data for the label pointed to by **pLabel**. If it finds it, the basename associated with it is copied to the buffer pointed to by **pBasename**. **pBasename** should point to a buffer of at least 9 bytes (maximum length of 8 plus 1 for the null).

DeskMate Technical Reference

Library Functions

dmmore_get_label (pLabel, pBasename)

char *pLabel;

char *pBasename;

dmmore_get_label returns the label associated with a basename after the error **DMMORE_BASENAME_NOT_UNIQUE** is returned from **dmmore_add_accessory**. The label is required to delete the accessory entry in the DMMORE.CFG environment file.

Input Parameters

pLabel is a pointer to a buffer to which the desired label will be copied. The buffer must be at least 16 bytes.

pBasename is a pointer to the basename to be searched for so that its associated label may be found.

Return Value

<int>

DMMORE_OK if the label has been copied to the buffer pointed to by **pLabel**.

DMMORE_FILE_NOT_IN_MEMORY if the DMMORE.CFG environment has not been loaded into memory. This call should only be made after an error, **DMMORE_BASENAME_NOT_UNIQUE**, has been returned from **dmmore_add_accessory**.

DMMORE_BASENAME_NOT_FOUND if the basename pointed to by **pBasename** does not match any basenames in the dmmore environment.

Special Notes

dmmore_get_label requires the application to link with DeskMate 03.05 or later libraries (DM.LIB or DMMED.LIB).

dmmore_get_label checks to see if the dmmore environment is currently in memory and if it is searches the dmmore environment data for the basename pointed to by **pBasename**. If it finds it, the Label associated with it is copied to the buffer pointed to by **pLabel**. **pLabel** should point to a buffer of at least 16 bytes (maximum length of 15 plus 1 for the null).

DeskMate Technical Reference

Library Functions

Structures/Defines

About Versions

```

/*-----*/
/*
/* DMGUF.H contains the About structure definition */
/*-----*/

/* struct for passing the about command the applications version */
struct appl_versionx
{
    char *papp_title;          /* title of the application */
    char *papp_name;          /* executable filename of the app */
    char *pversion;           /* version number of the application */
    char *pcopyright_one;     /* copyright */
    char *pcopyright_two;     /* copyright */
};
typedef struct appl_versionx APPL_VERSION;

```

*papp_title is a pointer to the application title string. Maximum of 38 characters.
 *papp_name is a pointer to the .PDM file name string. Maximum of 13 characters.
 *pversion is a pointer to the ASCII version number string. Maximum of 5 characters.
 *pcopyright_one is a pointer to the a copyright message. Maximum of 38 characters.
 *pcopyright_two is a pointer to a second copyright message. Maximum of 38 characters

Date Conversion/International Support

```

/*-----*/
/*
/* DMGUF.H contains the International structures and defines */
/*-----*/

/* International information
 * Information as it comes back from the DOS country information call
 */

typedef struct pdmintl_desc
{
    unsigned Date_fmt;          /* Date format */
    char Currency_sym[5];      /* Currency symbol, */
                                /* null terminated */
    char Thousand_sep[2];      /* 1000's separator, */
                                /* null terminated */
    char Decimal_sep[2];       /* Decimal separator, */
                                /* null terminated */
    char Date_sep[2];          /* Date separator, */
                                /* null terminated */
    char Time_sep[2];          /* Time separator, */
                                /* null terminated */
    char Currency_fmt;         /* Currency format flags */
    char Sig_decimals;         /* significant digits in currency */
    char Time_fmt;             /* Time format */
    char far *mono;            /* monospace routine entry point */
    char DList_sep[2];         /* Data list separator */
    unsigned int zeros[5];     /* Reserved. Do not use. */
} PDMINTL;

```

DeskMate Technical Reference Library Functions

```

USA DATEFMT          0
EUROPE DATEFMT       1
TIMEFMT 12           0
TIMEFMT 24           1
CURFMT 0             0      /* symbol precedes, no spaces */
CURFMT 1             1      /* symbol follows, no spaces */
CURFMT 2             2      /* symbol precedes, 1 spaces */
CURFMT 3             3      /* symbol follows, 1 spaces */
CURFMT 4             4      /* symbol replaces decimal separator */
DATE_SEPARATOR       '._'

```

```

/*-----*/
/*
/* DMMORE.H contains the More Accessory defines */
/*
/*-----*/

```

```

MAX_LABEL_LENGTH      15
MAX_BASENAME_LENGTH   8
MAX_NUM_ACCS          20
MAX_SIZE_DMMORE_DATA  ( ( MAX_LABEL_LENGTH + 1 + \
                        MAX_BASENAME_LENGTH + 1 ) \
                        * MAX_NUM_ACCS )

```

```

/* dmmore_add_accessory() and dmmore_delete_accessory() return codes*/
DMMORE_OK              0
DMMORE_OPEN_FILE_ERROR -1
DMMORE_NO_FILE_EXISTS  -2
DMMORE_CREATE_FILE_ERROR -3
DMMORE_LABEL_NOT_UNIQUE -4
DMMORE_BASENAME_NOT_UNIQUE -5
DMMORE_TOO_MANY_ACCESSORIES -6
DMMORE_INVALID_LABEL    -7
DMMORE_INVALID_BASENAME -8
DMMORE_LABEL_NOT_FOUND  -9
DMMORE_WRITE_FILE_ERROR -10
DMMORE_ADD_TO_FILE_ERROR -11
DMMORE_DUPLICATE_ENTRY  -12

```

DMMORE.H did not exist. I have created it and placed it in the DeskMate #include directory.

DeskMate Technical Reference

Library Functions

1989 DeskMate 03.03.0x ONLY

```
void about_versions ( pAppVersion )  
APPL_VERSION *pAppVersion;
```

about_versions should be used to display the "About" dialog box. The "About..." menu option should be the last item in the F2 File Menu and should be separated from the other items by a line.

Input Parameters

pAppVersion is a pointer to the APPL_VERSION structure. The APPL_VERSION structure is defined in dmguif.h.

Return Value

None.

Example

```
/* ABOUT DEFINITIONS */  
char AppnameAboutTitleStr[] = {"Appname"};  
char AppnameApplicationNameStr[] = {"APPNAME.PDM"};  
char AppnameVersionStr[] = {"01.00"};  
char AppnameCopyrightOneStr[] = {"DeskMate by Tandy"};  
char AppnameCopyrightTwoStr[] = {"Compliments of Tandy Corporation"};  
  
APPL_VERSION AppnameAPPL_VERSION =  
{  
    AppnameAboutTitleStr,  
    AppnameApplicationNameStr,  
    AppnameVersionStr,  
    AppnameCopyrightOneStr,  
    AppnameCopyrightTwoStr,  
};  
  
/* make the DeskMate library call to */  
/* display an ABOUT... dialog box */  
about_versions( &AppnameAPPL_VERSION );
```

DeskMate Technical Reference

Library Functions

cnvrt_to_julian (pDate, pBuf, pIntl)

char *pDate, *pBuf;
PDMINTL *pIntl;

cnvrt_to_julian provides a uniform date field input to the database manager.

Input Parameters

pDate is a pointer to null-terminated date string to be converted to julian format. The date may use a day/month/year field separator. The year may be two digits, 1900 is assumed century (so 01 is 1901, not 2001 as in DOS.) The day and month may be one or two digits. Validity checking is performed.

pBuf is a pointer to an 8 character buffer to hold null-terminated julian date string (YYYYDDD0).

pIntl is a pointer to 32-byte DOS country dependent information structure. Set to NULL for default USA date format. (MM-DD-YYYY)

Return Value

<int>

Zero (0) if a valid julian date has been placed in the buffer.

DM_ERROR if the given date string could not be interpreted as a valid date.

Special Notes

guf_bind_init must be made prior to making the **cnvrt_to_julian** call.

The intended use of **cnvrt_to_julian** and **cnvrt_to_std** is to provide a uniform date field input to the database, without putting the conversion and validation code into the database manager. To enter a date field in a database, the application should convert the date to julian and add as usual with a field type of 'D'. To display data field information, the application should retrieve the date field as normal, then call **cnvrt_to_std** to convert to standard DOS format before display.

See Also

get_intl()

DeskMate Technical Reference

Library Functions

cnvrt_to_std (pDate, pBuf, pIntl)
char *pDate, *pBuf;
PDMINTL *pIntl;

cnvrt_to_std provides standard format date from international julian format.

Input Parameters

pDate is a pointer to null-terminated date string to be converted to standard format. The first four characters are taken to be the year. The next three are assumed to be the julian day. Month and day are calculated and put in the provided buffer, separated by the appropriate character. The formatted date will have all fields padded with NULL as needed. **NO** validity checking is performed on the input julian date.

pBuf is a pointer to an 11 character buffer to hold null-terminated standard date string (MM-DD-YYYY0 for USA, DD-MM-YYYY for Europe). The separation character will vary from country to country based on information in the PDMINTL structure.

pIntl is a pointer to 32-byte DOS country dependent information structure. Set to NULL for default USA date format. (MM-DD-YYYY)

Return Value

<int>

Always zero (0) since no validity checking is done.

Special Notes

guf_bind_init must be made prior to making the **cnvrt_to_std** call.

See Also

get_intl()

DeskMate Technical Reference

Library Functions

int **compare_chars** (c1, c2)
char c1, c2;

comp_chars returns the difference between **c1** and **c2**.

Input Parameters

c1, **c2** are the two character values to be compared.

Return Value

Greater than zero if **c1** > **c2**.

Zero if **c1** = **c2**.

Less than zero if **c1** < **c2**.

Special Notes

If there is an international collating sequence table available (DOS 3.3 or greater), **c1** and **c2** will be changed to their collation values before subtracting.

compare_chars will make the international collating sequence translation if a collating sequence table is available. This happens for non-USA countries with DOS 3.3 or later. **compare_chars** allows the database to sort records in the correct order on international machines.

compare_chars is case sensitive, 'A' does not equal 'a'. If an application wants to do case insensitive compares, **comp_chars_ins** should be called.

Example

```
char c1 = 'A', c2 = 'Z';  
erc = compare_chars( c1, c2 );  
if ( erc == 0 ) /* characters are the same */  
;  
else if ( erc < 0 ) /* c1 is less than c2 - true for this example */  
;  
else /* erc > 0, c1 is greater than c2 */  
;  
;
```

See Also

comp_chars_ins()

DeskMate Technical Reference Library Functions

```
int comp_chars_ins ( c1, c2 )  
char c1, c2;
```

comp_chars_ins returns the difference between **c1** and **c2**.

Input Parameters

c1, **c2** are the two character values to be compared.

Return Value

Greater than zero if **c1** > **c2**.

Zero if **c1** = **c2**.

Less than zero if **c1** < **c2**.

Special Notes

comp_chars_ins will make the international collating sequence translation if a collating sequence table is available. This happens for non-USA countries with DOS 3.3 or later. **comp_chars_ins** allows the database to sort records in the correct order on international machines.

If there is an international collating sequence table available (DOS 3.3 or greater), the upper case equivalents of **c1** and **c2** will be changed to their collation values before subtracting.

This routine is NOT case sensitive. 'A' = 'a'.

Example

```
char c1 = 'A', c2 = 'a';  
  
erc = compare_chars_ins ( c1, c2 );  
if( erc == 0 ) /* characters are the same - true for this example */  
;  
else if( erc < 0 ) /* c1 is less than c2 */  
;  
else /* erc > 0 , c1 is greater than c2 */  
;  
;
```

See Also

to_upper()
comp_chars()

DeskMate Technical Reference

Library Functions

1989 DeskMate 03.03.0x ONLY

dmmore_add_accessory (pLabel, pBasename)

char *pLabel;

char *pBasename;

dmmore_add_accessory adds an accessory to the DMMORE.CFG configuration file which is used by the More accessory. The More accessory is used to run application defined accessories which are not included on the default F10 accessory menu.

Input Parameters

pLabel is a pointer to a string to be displayed in the More... accessory list box. The null terminated string may be from 1 to 15 characters long (excluding null).

pBasename is a pointer to a string which will be appended with ".ACC" to form an accessory filename to be executed when its respective **pLabel** string is selected from the More Accessories list box. The null terminated string may be from 1 to 8 characters long (excluding null).

Return Values

<int>

DMMORE_OK if the label and basename strings were successfully added to the dmmore environment in memory and to the DMMORE.CFG file on disk.

DMMORE_INVALID_LABEL if the label string is a null string or longer than the maximum of 15 characters.

DMMORE_INVALID_BASENAME if the basename string is a null string or longer than the maximum of 8 characters.

DMMORE_LABEL_NOT_UNIQUE if the label matches another label in the dmmore environment.

DMMORE_BASENAME_NOT_UNIQUE if the basename matches another basename in the dmmore environment.

DMMORE_DUPLICATE_ENTRY if the label and basename both match another entry in the file. A duplicate entry is not allowed.

DMMORE_OPEN_FILE_ERROR if an error occurred trying to open the file (such as the file was not found) or there was not enough environment space to load in the data of the DMMORE.CFG environment file.

DMMORE_CREATE_FILE_ERROR if an error occurred trying to create a new DMMORE.CFG file or an existing DMMORE.CFG file was found when trying to create the file and there was not enough memory to load the data of the file.

DMMORE_TOO_MANY_ACCESSORIES if there are already 20 entries (the maximum) in the dmmore environment.

DMMORE_ADD_TO_FILE_ERROR if there was not sufficient memory to add the entry to the dmmore environment.

DMMORE_WRITE_FILE_ERROR if an error occurred writing the updated dmmore environment to the DMMORE.CFG file.

DeskMate Technical Reference

Library Functions

Special Notes

dmmore_add_accessory checks to see if the dmmore environment is currently in memory and if it is adds the new entry to the list and updates the DMMORE.CFG file on disk (prompting the user for the disk if file not found and DMCONFIG set to a floppy drive). If the dmmore environment is not in memory, it checks a status bit in the CSR environment to see if the dmmore environment was previously created and if so, opens the DMMORE.CFG file (prompting the user for the disk if file not found and DMCONFIG set to a floppy drive), adds the new entry and updates the DMMORE.CFG file on disk. If the dmmore environment had not been previously created, the call creates the file (prompting the user for the disk on which to create the file if DMCONFIG set to a floppy drive), adds the new entry, updates the DMMORE.CFG file on the disk, and updates status bit in the CSR environment to register the creation of the DMMORE.CFG file.

DeskMate Technical Reference

Library Functions

1989 DeskMate 03.03.0x ONLY

dmmore_delete_accessory (pLabel)
char *pLabel;

dmmore_delete_accessory deletes an accessory from the DMMORE.CFG configuration file which is used by the More accessory. The More accessory is used to run application defined accessories which are not included on the default F10 accessory menu.

Input Parameters

pLabel is a pointer to a string to be displayed in the More... accessory list box. The null terminated string should be from 1 to 15 characters long (excluding null) and match an existing entry in the dmmore environment.

Return Value

<int>

DMMORE_OK if the label and its respective basename were successfully deleted from the dmmore environment in memory and the updated environment saved to the DMMORE.CFG file.

DMMORE_OPEN_FILE_ERROR if an error occurred trying to open the file (such as the file was not found) or there was not enough environment space to load in the data of the DMMORE.CFG environment file.

DMMORE_NO_FILE_EXISTS if the dmmore environment is not currently in memory and a status bit in the CSR environment signifies that no dmmore environment was previously created.

DMMORE_LABEL_NOT_FOUND if the label does not match any labels in the dmmore environment.

DMMORE_WRITE_FILE_ERROR if an error occurred writing the updated dmmore environment to the DMMORE.CFG file.

Special Notes

dmmore_delete_accessory checks to see if the dmmore environment is currently in memory and if it is deletes the label and its respective basename from the list and updates the DMMORE.CFG file on disk (prompting the user for the disk if file not found and DMCONFIG set to a floppy drive). If the dmmore environment is not in memory, it checks a status bit in the CSR environment to see if the dmmore environment was previously created and if so, opens the DMMORE.CFG file (prompting the user for the disk if file not found and DMCONFIG set to a floppy drive), deletes label and its respective basename and updates the DMMORE.CFG file on disk. If the dmmore environment had not been previously created, the call returns with the error, DMMORE_NO_FILE_EXISTS.

DeskMate Technical Reference Library Functions

1989 DeskMate 03.03.0x ONLY

get_ds ()

get_ds returns the current data segment (value of DS).

Input Parameters

None.

Return Value

Current data segment

Example

```
help_notification.data_seg = get_ds();
```

See Also

hlp_notify()

This function does not appear to work. Use an _asm statement to get DS.

DeskMate Technical Reference Library Functions

```
int get_intl ( pIntl )  
PDMINTL      *pIntl;
```

get_intl fills in the user international information call according to the DOS 38h function call.

Input Parameters

pIntl is a pointer to a PDMINTL structure.

Return Value

Zero (0) if successful.

DM_ERROR with dmerrno set to the DOS error value if unsuccessful.

Special Notes

The PDMINTL structure is defined in dmguif.h.

get_intl must have been made before passing an international structure pointer to **cnvrt_to_std** or **cnvrt_to_julian**.

Example

```
PDMINTL Intl;  
  
erc = get_intl ( &Intl );
```

See Also

cnvrt_to_std()
cnvrt_to_julian()

DeskMate Technical Reference

Library Functions

```
int get_month_strings ( pMonths )  
char *pMonths;
```

get_month_strings will return in the user's buffer a 37 character string which contains three characters abbreviations for the months of the year. The string will be terminated by a single NULL.

Input Parameters

pMonths is a pointer to a 37 byte data area to be filled in.

Return Value

None.

Special Notes

get_month_strings is contained in the GUF resource, therefore **guf_bind_init** should be made before **get_month_strings** may be used.

Example

```
char Months[37];  
  
get_month_strings ( &Months );
```

DeskMate Technical Reference Library Functions

PageSetupDialog ()

PageSetupDialog allows the application to display the standard DeskMate Page Setup dialog box, without having to run the page setup accessory.

Input Parameters

None.

Return Value

Zero if the accessory failed to load.

RETURN_KEY (000Dh) if the OK push button was selected from the accessory.

ESC_KEY (001B) if the CANCEL push button was selected from the accessory.

Special Notes

PageSetupDialog requires the application to link with DeskMate 03.03 or later libraries (DM.LIB or DMMED.LIB).

PageSetupDialog can be used instead of **fil_menu_page** which does run the accessory.

See Also

fil_menu_page()

DeskMate Technical Reference

Library Functions

str_upper (pStr)
char *pStr;

str_upper changes each character in the given NULL-terminated string to it's upper case equivalent if it has one, including international characters above 7fh.

Input Parameters

pStr is a pointer to the NULL -terminated string to be converted.

Return Value

None.

Special Notes

str_upper will make the international information case mapping routine call for characters values above 7fh.

Characters without upper case equivalents are unchanged.

Example

```
/* \201 is the u with two dots over it */  
char str[] = { \201, 'a', 'z', '\0'};  
  
/* the result is */  
/* str[0] == value \232, U with two dots over it */  
/* str[1] == 'A', str[2] == 'Z' */  
str_upper ( str );
```

See Also

to_upper()

DeskMate Technical Reference

Library Functions

```
unsigned char to_upper ( ch )  
unsigned char      ch;
```

to_upper changes the given character to it's upper case equivalent if it has one, including international characters above 7fh.

Input Parameters

ch is the character to convert.

Return Value

<unsigned char>
The character given or upper case equivalent.

Special Notes

to_upper will make the international information case mapping routine call for character values above 7fh.
Characters without upper case equivalents are unchanged.

Example

```
char ch = \201; /* lower case u with two dots over it */  
ch = to_upper ( ch ); /* ch has value \232, U with two dots over it */
```

See Also

str_upper()

DeskMate Technical Reference

Library Functions

waitloop (Time)
int Time;

waitloop allows an application to pause for a given amount of time.

Input Parameters

Time has the number of seconds to pause in the high byte, hundredths of seconds to pause in the low byte. Seconds must be less than 60.

Return Value

None.

Menu Bar Manager

"Menubar Manager"

Table of Contents

General Description/Notes	17- 1
Structures/Defines	17- 2
mb_add_alarmAdds an alarm to the main menu bar.....	17- 6
mb_delete_alarm .Deletes an alarm from the menu bar.....	17- 7
mb_disableDisables auto processing of the menu bar.....	17- 8
mb_drawDraws the menu bar/registers menu accels.....	17- 9
mb_enableEnables auto processing of the menu bar.....	17-10
mb_eraseDeactivates the menu bar.....	17-11
mb_get_status ...Gets the number of the selected menu	17-12

General Description/Notes

This section refers to the Main Menu Bar. Also see component section to run the menu bar as a component.

The menu bar consists of a group of menu buttons. Each menu is a list of items or commands from which the user chooses. Every menu has a menu name which states the purpose of the menu as clearly as possible. DeskMate automatically assigns each menu a function key accelerator. Applications which access data files should have a file menu for file operations, and those applications which interface with the clipboard should have an edit menu. These two menus should be the first two on the menu bar, therefore their accelerator function keys are F2 and F3. The F1 key is reserved for the help function. The F9 key is reserved for the message menu. The F10 key is reserved for the accessory menu. Therefore there are seven menus available to the application.

Menu Bar and Dialog Box Return Codes

Return codes for Menu Bars and return codes for Dialog Boxes should always be unique. For example define menu bar return codes to be APP_TAG + 100, APP_TAG + 101, APP_TAG + 102, etc, and dialog box return codes to be APP_TAG + 1, APP_TAG + 2, APP_TAG + 3, etc.

Calculating the Size of a Menu Pull Down

Below is a formula to determine the maximum size of a menu pull down:

$$\begin{aligned} & ((\text{number of menu groups} - 1) + (\text{number of menu items})) \\ & * (\text{number of characters in longest menu item} + 6) \leq 363 \end{aligned}$$

To get the Y world coordinate of the pixel below the menubar use the following formula:

```
yorg = vid_next_nwcy( vid_wcy_to_nwcy( (MB_YORG) + MB_YEXT) - 1 );
```

DeskMate Technical Reference

Menu Bar Manager

Structures/Defines

```

/*-----*/
/*
/* CSRCMPS.H contains the Menu bar structures and defines */
/*-----*/

/* Menu bar Component */
struct menuitem_defn
{
    char          type;          /* item type */
    char          bEnabled;      /* enable (grayed) flag for item */
    char          bChecked;      /* check flag */
    unsigned int  accel;         /* accelerator for item */
    unsigned int  return_code;   /* return code for item */
    char          group;        /* group number of item */
    char          pattern;       /* pattern to use as the item */
    unsigned char *pString;      /* pointer to item name string */
};
typedef struct menuitem_defn MENUITEM;

```

MENUITEM.type:

Defines the type of menuitem (MB_STRING or MB_PATTERN).

```

/* Item types */
MB_STRING      0    /* item is a string */
MB_PATTERN     1    /* item is a pattern */

```

MENUITEM.bEnabled:

Enable flag for the menuitem (ENABLED or DISABLED).

MENUITEM.bChecked:

Indicates whether or not the menuitem is checked or unchecked (MB_CHECKED or MB_UNCHECKED).

```

/* Item check state */
MB_CHECKED      SELECTED
MB_UNCHECKED    DESELECTED

```

MENUITEM.accel:

Keyboard accelerator for this menuitem (NO_ACCEL).

```

/* Accelerators */
NO_ACCEL        -1    /* no accelerator */
MB_ACCEL        ALT_SPACE_KEY

```

MENUITEM.return_code:

Return code for the menuitem.

MENUITEM.group:

Group number of the menuitem (MB_GROUP1 to MB_GROUP9). The group numbers **MUST** be sequential and **MUST** begin with MB_GROUP1 for each menu. Changing the group number automatically draws a separator line in the menuitem list.

```

/* Groups */
MB_GROUP1      0
MB_GROUP2      1
MB_GROUP3      2
MB_GROUP4      3
MB_GROUP5      4
MB_GROUP6      5

```

DeskMate Technical Reference Menu Bar Manager

MB_GROUP7 6
MB_GROUP8 7
MB_GROUP9 8

MENUITEM.pattern:

Pattern number of the menuitem (PATTERN1 to PATTERN16). Pattern is only valid if the type element is MB_PATTERN.

MENUITEM.pString:

Pointer to the menuitem string. pString is valid only if the type element is MB_STRING.

```
struct menu_defn
{
    int          xext;          /* x extent of menu */
    unsigned char *pString;     /* pointer to button name string */
    char         nItems;        /* number of items in the menu */
    MENUITEM     *pItems;       /* pointer to MENUITEM structures */
};
typedef struct menu_defn MENU;
```

MENU.xext:

World unit x extent of the menu.

MENU.pString:

Pointer to menubutton label string.

MENU.nItems:

Number of items in the menu.

MENU.pItems:

Pointer to the MENUITEM structures for the menu.

```
struct menubar_defn
{
    CMP_HEADER header;
    char       bFlags;          /* arrow and top line flags */
    char       bRedraw;         /* visible redraw flag */
    char       nMenus;          /* number of menus on the menu bar */
    MENU       *pMenus;         /* pointer to MENU structures */
    char       pattern;         /* background pattern of the menu bar */
    char       bStatus;         /* NOT USED */
};
typedef struct menubar_defn MENUBAR;
```

MENUBAR.header:

This header is a header structure which provides information to the component manager. Refer to the CMP_HEADER documentation for a further discussion of component headers.

```
/* Header maprect */
MB_XORG     0                 /* default x origin of a menu bar */
MB_YORG     290               /* default y origin of a menu bar */
MB_XEXT     80*CHAR_XEXT      /* default x extent of a menu bar */
MB_YEXT     345               /* y extent of menu bars */
```

MENUBAR.bFlags:

Indicates which buttons are to be available on the menu bar as defined below. bFlags shows what kind of icons are to appear at the edge of the menu bar. They are defined in CSRCMPS.H.

DeskMate Technical Reference Menu Bar Manager

For example: MB_TANDY + MB_ALL_ARROWS means the Tandy Icon (Accessories Menu) and all four of the arrow icons will show on the menu bar line.

```
/* Flag masks */
MB_PLAIN      0 /* no Tandy, help or arrow buttons */
MB_HELP_BUTTON 1 /* help button */
MB_RIGHT_ARROW 2 /* right arrow button */
MB_LEFT_ARROW 4 /* left arrow button */
MB_DOWN_ARROW 8 /* down arrow button */
MB_UP_ARROW   16 /* up arrow button */
MB_TANDY      32 /* Accessories F10 button */
MB_ALARM      64 /* Alarm F9 button */
MB_VERT_ARROWS MB_UP_ARROW+MB_DOWN_ARROW
MB_HORZ_ARROWS MB_LEFT_ARROW+MB_RIGHT_ARROW
MB_ALL_ARROWS  MB_VERT_ARROWS+MB_HORZ_ARROWS
```

MB_HELP_BUTTON should not be used. No help button appears, and DeskMate 3.02 can crash if you specify one.

MENUBAR.bRedraw:

Indicates whether or not to redraw the menubuttons on **cmp_draw** or **mb_draw** (SELECTED or DESELECTED). This element must be SELECTED the first time **mb_draw** is used, but may be deselected on subsequent calls. Also if the application only wishes to change the enable or checked flags of the menu items, **cmp_open** will select this element and **cmp_draw** will deselect it.

```
/* Redraw flags */
MB_REDRAW      SELECTED /* redraw menu bar */
MB_NO_REDRAW    DESELECTED /* only redefine enabled/disabled items */
```

MENUBAR.nMenus:

Number of labeled menubutton on the menu bar.

MENUBAR.pMenus:

Pointer to the MENU structures for the menu bar.

MENUBAR.pattern:

Background pattern number of the menu bar (PATTERN1 to PATTERN20).

MENUBAR.bStatus:

For internal use only.

```
/* World coordinate of the last default main menu bar pixel */
MB_BOTTOM_YORG MB_YORG+MB_YEXT-1
```

```
/* mb add alarm return codes */
MB_NOT_NOTIFIED 0x01 /* alarm added but User not notified */
MB_ALARM_EXISTS 0x02 /* alarm exists, not added again */
```

```
/* Main Menu bar Manager ALARM structure */
```

```
struct alarm_defn
{
    char *pType; /* <=16 char ASCIIIZ string for menuitem */
    char *pMessage; /* <=120 char ASCIIIZ string for message */
    char bRemove; /* remove alarm on User selection flag */
    int (far *pSelected)(); /* far call upon User selection address */
};
```

```
typedef struct alarm_defn ALARM;
```

The ALARM structure is defined as follows for this service.

```
typedef struct alarm_defn ALARM;
```

DeskMate Technical Reference

Menu Bar Manager

ALARM.pType

A pointer to an ASCIIZ string of 16 characters or less to be used as the menu item for the alarm within the alarm menu. If a message does exist then an ellipse is added to the pType string.

ALARM.pMessage

A pointer to an ASCIIZ string of 120 characters or less to be used as the alarm message. If the pSelected element of this structure is not a null pointer, then this element is ignored. This string must also meet the requirements for a message string as limited by the Message Box Manager. If this element is a null pointer and pSelected element is a null pointer, the Alarm Manager will do nothing upon selection of the alarm item.

ALARM.bRemove

Indicates whether the alarm item be removed or remain in the alarm menu. If bRemove is equal to DESELECTED then the element is to be removed. If bRemove is equal to SELECTED the alarm item remains.

ALARM.pSelected

A far pointer to a user defined routine to be called when the alarm item is selected by the user. If this is a pointer to a null string, then the Alarm Manager will use the pMessage element to display a message box upon user selection. (No parameters are passed to the specified routine when the alarm item is selected, therefore each alarm added may require a separate user selection routine.)

DeskMate Technical Reference

Menu Bar Manager

mb_add_alarm (pALARM)
ALARM *pALARM;

mb_add_alarm notifies the User that some type of alarm event has occurred that requires their attention.

Input Parameters

pALARM points to an ALARM structure.

Return Value

MB_NOT_NOTIFIED if the alarm added but the User was not notified.

MB_ALARM_EXISTS if the alarm already exists.

CSR_ERROR if 8 alarm items already exist in the alarm menu.

Special Notes

Once an alarm is added the alarm button of the main menu bar (if available) will begin a flash cycle to get the attention of the user. The ! F9 within the alarm button will flash three times, wait about 15 seconds and continue that cycle until the alarm menu is **ACCESSED**. The button will stay inverted until the user pulls down the alarm at which time the flash cycle will terminate, and revert to it's original attribute. It is not required that the user select an item to terminate the flash cycle. The flash cycle will only be active when the main menu bar has been drawn with **mb_draw**.

See Also

mb_delete_alarm()

The alarm is defined in CSRBASE.H.

DeskMate Technical Reference

Menu Bar Manager

mb_delete_alarm (pALARM)
ALARM *pALARM;

mb_delete_alarm removes an alarm menu item from the alarm menu.

Input Parameters

pALARM points to an ALARM structure.

Return Value

CSR_ERROR if the alarm items does not exist in the alarm menu.

Special Notes

The ALARM structure is defined as follows for this service.

```
/* Main Menu bar Manager ALARM structure */
struct alarm_defn
{
    char *pType;                /* This string must match exactly
                                /* the string used to add the
                                /* alarm */
    char *pMessage;             /* This element is ignored */
    char bRemove;               /* This element is ignored */
    int (far *pSelected) ();    /* This element is ignored */
};
typedef struct alarm_defn ALARM;
```

See Also

mb_add_alarm()
The alarm is defined in CSRBASE.H.

DeskMate Technical Reference Menu Bar Manager

mb_disable ()

mb_disable disables autoprocessing of the main menu bar.

Input Parameters

None.

Return Value

None.

Special Notes

The main menu bar is the menu bar created with the **mb_draw** service. **mb_draw** automatically enables autoprocessing of the defined main menu bar.

See Also

mb_enable()
mb_draw()
mb_erase()

DeskMate Technical Reference

Menu Bar Manager

mb_draw (pMENUBAR)
MENUBAR *pMENUBAR;

mb_draw draws the specified menu bar as the main menu bar and registers the menu with **event_read**.

Input Parameters

pMENUBAR is a pointer to a menu bar structure.

Return Value

CSR_ERROR if the menu bar is not created.

Special Notes

Once **mb_draw** is made by an application, all processing of the menu bar is done automatically by the Event Manager. The menu bar structure may be dynamically altered by the application by making subsequent calls to **mb_draw**. The original call to **mb_draw** must be made from the window that the application considers the parent window of the menu bar. However, once a main menu bar has been registered, subsequent calls to **mb_draw** may be made from any window.

mb_erase does not clear the menu bar from the screen. **mb_erase** must be called if menu options are added to or removed from a menu during program execution.

mb_draw is also used to reactivate accelerators on previously grayed menu items. For speed considerations, applications should make sure the **bRedraw** flag is not set when making subsequent calls to **mb_draw**, unless the menu bar needs to be "physically" redrawn.

The following icon codes may be used to specify which icons appear at the right edge of the menu bar:

MB_PLAIN	0	/* no Tandy, help or arrow buttons */
MB_HELP_BUTTON	1	/* help button */
MB_RIGHT_ARROW	2	/* right arrow button */
MB_LEFT_ARROW	4	/* left arrow button */
MB_DOWN_ARROW	8	/* down arrow button */
MB_UP_ARROW	16	/* up arrow button */
MB_TANDY	32	/* Accessories F10 button */
MB_ALARM	64	/* Alarm F9 button */
MB_VERT_ARROWS	MB_UP_ARROW+MB_DOWN_ARROW	
MB_HORZ_ARROWS	MB_LEFT_ARROW+MB_RIGHT_ARROW	
MB_ALL_ARROWS	MB_VERT_ARROWS+MB_HORZ_ARROWS	

Menu items may be shaded or unshaded (**DISABLED** or **ENABLED**).

Menu items may be checked or unchecked (**CHECKED** or **UNCHECKED**).

Menu items may have a line appear below them, ending the menu item group.

Therefore, the structure item group is provided to indicate which menu item group a menu item belongs to. If a menu has only one menu item group (no separating lines), then the application would set each item's group to **GROUP1**. If there was a second group, each of the items in the group would have a group set to **GROUP2**. A menu may have at most, 9 groups (**GROUP1** through **GROUP9**).

DeskMate Technical Reference

Menu Bar Manager

mb_enable ()

mb_enable enables autoprocessing of the main menu bar.

Input Parameters

None.

Return Value

None.

Special Notes

The main menu bar is the menu bar created with the **mb_draw** service. **mb_draw** automatically enables autoprocessing of the defined main menu bar. This service is only required when a previous **mb_disable** call has been made.

See Also

mb_disable()
mb_draw()
mb_erase()

DeskMate Technical Reference

Menu Bar Manager

mb_erase (pMENUBAR)
MENUBAR *pMENUBAR;

mb_erase removes the current menu bar, but does not clear the menu bar from the screen.

Input Parameters

pMENUBAR is a pointer to a MENUBAR structure.

Return Value

None.

Special Notes

The Event Manager will no longer process a main menu bar.

mb_erase must be called if menu options are added to or removed from a menu during program execution.

*This is wrong. mb_erase()
does not take any parameters.*

DeskMate Technical Reference Menu Bar Manager

1989 DeskMate 03.03.0x ONLY

mb_get_status ()

mb_get_status returns the number of the currently selected menu and menu option.

Input Parameters

None.

Return Value

<int>

Low order byte:

CSR_ERROR if no menu option is selected.

One (1) if the first menu option is selected, etc.

High order byte:

CSR_ERROR if no menu is selected.

Two (2) if the F2 menu is selected.

Three (3) if the F3 menu is selected, etc.

Message Box Manager

"Message Box Manager"

Table of Contents

General Description/Notes	18- 1
Structures/Defines	18- 2
msg_drawDisplays a message box.....	18- 3
msg_runDisplays and processes a message box.....	18- 4

DeskMate Technical Reference Message Box Manager

General Description/Notes

A message box is a simplified dialog box. A message box contains a title, a static string message, and push buttons. Message boxes inform the user some action has occurred, such as displaying an error message, or to ask the user for confirmation of an action.

DeskMate Technical Reference Message Box Manager

Structures/Defines

```
/*-----*/
/*
/* CSRCMPS.H contains the Message box structure and defines */
/*
/*-----*/

struct msgbox_defn
{
    char      btn_combo;      /* specifies button combination */
    unsigned char *pString;    /* a pointer to title string */
    unsigned char *pMessage;   /* a pointer to message string */
};
typedef struct msgbox_defn MSGBOX;
```

MSGBOX.btn_combo

The button combination for the message box as defined below.

MSGBOX.pString

Pointer to the title string for the message box.

MSGBOX.pMessage

Pointer to message for the message box. The string will be automatically word wrapped by the Message Component. The string must fit on three 36-character word-wrapped lines. If the string exceeds three lines then the extra lines will be truncated.

```
/* Button combinations */
MSG_COMBO_OK          0    /* OK button only */
MSG_COMBO_OK_CAN      1    /* OK and CANCEL buttons */
MSG_COMBO_YES_NO_CAN  2    /* YES, NO and CANCEL buttons */
MSG_COMBO_CANCEL      12   /* CANCEL button only */
MSG_COMBO_RETRY_CAN   13   /* RETRY and CANCEL buttons */
MSG_COMBO_YES_NO      14   /* YES and NO buttons */

/* Button return codes */
MSG_OK                MSG_TAG+1
MSG_CANCEL            MSG_TAG+2
MSG_YES               MSG_TAG+3
MSG_NO                MSG_TAG+4
MSG_RETRY             MSG_TAG+11

/* coordinates of single push button in a message box */
MSG_HS_XORG           36*CHAR_XEXT
MSG_HS_YORG           14*CHAR_YEXT
MSG_HS_XEXT           8*CHAR_XEXT

/* Message box save rectangle */
MSG_SAVE_XORG         19*CHAR_XEXT
MSG_SAVE_YORG         8*CHAR_YEXT
MSG_SAVE_XEXT         42*CHAR_XEXT
MSG_SAVE_YEXT         9*CHAR_YEXT
```

DeskMate Technical Reference

Message Box Manager

msg_draw (pMSGBOX)
MSGBOX *pMSGBOX;

msg_draw displays a message box.

Input Parameters

pMSGBOX is a pointer to a MSGBOX structure.

Return Value

None.

Special Notes

The usable width of the message box is 36 characters. The message appears in the box as three word wrapped lines.

msg_draw does **NOT** restore the background when finished.

The message box string length is limited to 108 characters.

The following are the message box push button combinations that are supported:

MSG_COMBO_OK	0	/* OK button only */
MSG_COMBO_OK_CAN	1	/* OK and CANCEL buttons */
MSG_COMBO_YES_NO_CAN	2	/* YES, NO and CANCEL buttons */
MSG_COMBO_CANCEL	12	/* CANCEL button only */
MSG_COMBO_RETRY_CAN	13	/* RETRY and CANCEL buttons */
MSG_COMBO_YES_NO	14	/* YES and NO buttons */

See Also

msg_run()
MSGBOX structure definition

DeskMate Technical Reference

Message Box Manager

msg_run (pMSGBOX)
MSGBOX *pMSGBOX;

msg_run displays and processes a message box.

Input Parameters

pMSGBOX is a pointer to a MSGBOX structure.

Return Value

<int>

The return code of the message box push button chosen by the user.

Special Notes

The usable width of the message box is 36 characters. The message appears in the box as three word wrapped lines.

msg_run always restores the background when finished.

The message box string length is limited to 108 characters.

The following are the supported message box push button combinations:

MSG_COMBO_OK	0	/* OK button only */
MSG_COMBO_OK_CAN	1	/* OK and CANCEL buttons */
MSG_COMBO_YES_NO_CAN	2	/* YES, NO and CANCEL buttons */
MSG_COMBO_CANCEL	12	/* CANCEL button only */
MSG_COMBO_RETRY_CAN	13	/* RETRY and CANCEL buttons */
MSG_COMBO_YES_NO	14	/* YES and NO buttons */

The following are the codes which can be returned from a message box:

MSG_OK	MSG_TAG+1
MSG_CANCEL	MSG_TAG+2
MSG_YES	MSG_TAG+3
MSG_NO	MSG_TAG+4
MSG_RETRY	MSG_TAG+11

See Also

msg_draw()

Mouse Manager

"Mouse Manager"

Table of Contents

General Description/Notes	19- 1
Structures/Defines	19- 2
ms_add_region	Defines mouse pointer to a region.....19- 3
ms_default_pointer	Specifies default arrow pointer.....19- 4
ms_define_pointer	Define a custom mouse pointer.....19- 5
ms_delete_region	Deletes a mouse pointer region.....19- 7
ms_disable_driver	Disables the mouse driver.....19- 8
ms_disable_regions	Deletes all mouse pointer regions.....19- 9
ms_draw_pointer	Turn on the mouse pointer.....19-10
ms_enable_driver	Initializes the mouse driver.....19-11
ms_enable_regions	Enables all mouse pointer regions.....19-12
ms_erase_pointer	Turns off the mouse pointer.....19-13
ms_get_status	Gets the current mouse status.....19-14

DeskMate Technical Reference

Mouse Manager

General Description/Notes

The mouse manager maintains and updates the mouse pointer for the application, as well as performing internal functions for the event manager. The application can control installation of the mouse drivers, the usage of the mouse pointer, definition of the mouse pointer, and mouse pointer regions on the display.

Internally, the mouse manager maintains the mouse pointer on the display and detects changes in the button state and assimilates that data into mouse events for the event manager. The mouse manager does not differentiate buttons on those pointing devices with multiple buttons. They are interpreted as a button not as a left/right/middle button. The mouse manager responds only to the outside buttons on those pointing devices with multiple buttons.

The currently supported pointing devices are:

- Tandy 1000 Joystick
- Serial Mouse
- PS2 compatible mouse ports.

The mouse driver installed at initialization is determined by DMCSR.CFG.

Some versions of certain 1988 DeskMate applications did not delete their mouse regions prior to exiting. This causes mouse pointer regions to remain defined for subsequent applications. The problem is not exhibited unless a subsequent application enables the mouse pointer regions. To ensure that all regions are removed for your application, refer to **ms_add_region** before adding any regions, also, your application should delete any mouse pointer regions prior to exiting.

DeskMate Technical Reference

Mouse Manager

Structures/Defines

```

/*-----*/
/*
/* CSRBASE.H contains the Mouse structures and defines */
/*-----*/

/* Mouse status */
struct mouse_stat_defn
{
    char    bDriver;        /* enable state of driver */
    char    bPointer;       /* on/off state of pointer */
    char    bRegions;       /* enable state of pointer regions */
    char    button_state;   /* status of the mouse buttons */
    int     pointer_x;      /* mouse pointer wc x position */
    int     pointer_y;      /* mouse pointer wc y position */
};
typedef struct mouse_stat_defn MOUSE_STAT;

MOUSE_STAT.bDriver ENABLED or DISABLED.
MOUSE_STAT.bPointer ENABLED or DISABLED.
MOUSE_STAT.bRegions ENABLED or DISABLED.
MOUSE_STAT.button_state BUTTON UP or BUTTON_DOWN.
MOUSE_STAT.pointer_x device x coordinate.
MOUSE_STAT.pointer_y device y coordinate.

MS_BUTTON_OPEN      0    /* mouse button is up */
MS_BUTTON_CLOSED    1    /* mouse button is closed */

/* Mouse pointer regions */
struct ptr_region_defn
{
    char    id;             /* application defined id for region */
    MAPRECT maprect;
    char    focal_x;        /* mouse hot spot x */
    char    focal_y;        /* mouse hot spot y */
    int     *pPointer;      /* pointer to mouse pointer definition */
};
typedef struct ptr_region_defn PTR_REGION;

PTR_REGION.id        char    - application defined ID for the region.
PTR_REGION.maprect;  int     - world coordinate x origin of the
                        pointer region relative to the
                        base screen.
                        MAPRECT.xorg
                        int     - world coordinate y origin of the
                        pointer region relative to the
                        base screen.
                        MAPRECT.yorg
                        int     - world unit x extent of the pointer
                        region.
                        MAPRECT.xext
                        int     - world unit y extent of the pointer
                        region.
                        MAPRECT.yext
PTR_REGION.focal_x   char    - mouse hot spot x coordinate relative to
                        the upper-left corner of the pointer
                        definition.
PTR_REGION.focal_y   char    - mouse hot spot y coordinate relative to
                        the upper-left corner of the pointer
                        definition.
PTR_REGION.pPointer   int     - pointer to the mouse pointer definition
                        and mask.

```

DeskMate Technical Reference

Mouse Manager

ms_add_region (pPTR_REGION)
PTR_REGION *pPTR_REGION;

ms_add_region defines a specific mouse pointer for a specific area of the display.

Input Parameters

pPTR_REGION is a pointer to a **PTR_REGION** region structure.

Return Value

<int>

CSR_ERROR if the mouse is not available.

Special Notes

The pointer region is always relative to the base window.

The created pointer region is not accessed with a handle, but instead with an ID that is supplied by the application in the pointer region structure.

Once **ms_add_region** is called, the mouse pointer will automatically change to the specified pointer image when the mouse pointer is moved into the specified area of the display. Some CSR services, such as **dlg_run** and **msg_run**, will disable this feature while executing.

The priority of the regions is determined by the order that they are defined. If two regions overlap, the most recently defined pointer region will determine the pointer image.

Some versions of certain 1988 DeskMate applications did not delete their mouse regions prior to exiting. This causes mouse pointer regions to remain defined for subsequent applications. The problem is not exhibited unless a subsequent application enables the mouse pointer regions. To ensure that all regions are removed for your application, use the following lines of code before adding your first region:

```
int count; id_count;  
  
for ( count = 1; count <= 3; count++ )  
    for ( id_count = 0; id_count <= 255; id_count++ )  
        ms_delete_region( ( char ) id_count );
```

Also, your application should delete any mouse pointer regions prior to exiting.

Example

```
int ReturnCode;  
  
/* add a region */  
ReturnCode = ms_add_region( PTR_REGION );
```

See Also

dlg_run() in the dialog box manager section.

msg_run() in the message box manager section.

DeskMate Technical Reference

Mouse Manager

ms_default_pointer ()

ms_default_pointer specifies that the mouse pointer is defined to be the default arrow pointer.

Input Parameters

None.

Return Value

<int>

CSR_ERROR if the mouse is not available.

Example

```
int ReturnCode;  
  
/* define the mouse pointer to be the default arrow */  
ReturnCode = ms_default_pointer();
```

DeskMate Technical Reference

Mouse Manager

ms_define_pointer (FocalX, FocalY, plmage)

char FocalX;
char FocalY;
char *plmage;

ms_define_pointer defines the mouse pointer.

Input Parameters

FocalX is the world coordinate x of the focal point of the defined mouse cursor relative to the origin of the pointer image.

FocalY is the world coordinate y of the focal point of the defined mouse cursor relative to the origin of the pointer image.

plmage is a pointer to two image tables defining the mouse cursor (refer to the digi-mouse reference manual for more detail on the definition tables).

Return Value

<int>

CSR_ERROR if the mouse is not available.

Example

```
int    ReturnCode;
char   MsFocalX;
char   MsFocalY;
char   *pMsImage;

/* define the mouse pointer */
ReturnCode = ms_define_pointer( MsFocalX, MsFocalY, pMsImage );

/* This is a sample showing how to define a new mouse pointer */
/* or a defined text cursor */
/* The sample uses the masks for the standard mouse pointer */
int cursorDef[] =
{
    0x3FFF,      /* line 1 start of and_mask */
    0x1FFF,      /* line 2 */
    0x0FFF,      /* line 3 */
    0x07FF,      /* line 4 */
    0x03FF,      /* line 5 */
    0x01FF,      /* line 6 */
    0x00FF,      /* line 7 */
    0x007F,      /* line 8 */
    0x003F,      /* line 9 */
    0x001F,      /* line 10 */
    0x000F,      /* line 11 */
    0x000F,      /* line 12 */
    0x007F,      /* line 13 */
    0x183F,      /* line 14 */
    0xFC1F,      /* line 15 */
    0xFE1F,      /* line 16 end of and mask */
    0x0000,      /* line 1 start of xor_mask */
    0x4000,      /* line 2 */
    0x6000,      /* line 3 */
    0x7000,      /* line 4 */
}
```

DeskMate Technical Reference

Mouse Manager

```
0x7800, /* line 5 */
0x7C00, /* line 6 */
0x7E00, /* line 7 */
0x7F00, /* line 8 */
0x7F80, /* line 9 */
0x7FC0, /* line 10 */
0x7FE0, /* line 11 */
0xFF00, /* line 12 */
0x6700, /* line 13 */
0x3800, /* line 14 */
0x1C00, /* line 15 */
0x0000, /* line 16 end of xor_mask */
};
```

DeskMate Technical Reference

Mouse Manager

ms_delete_region (ID)

char ID;

ms_delete_region terminates the specified pointer region.

Input Parameters

ID is the ID of the specified pointer region that the application specified in the PTR_REGION structure that was used to define the pointer region.

Return Value

<int>

CSR_ERROR if the mouse is not available.

Special Notes

Some versions of certain 1988 DeskMate applications did not delete their mouse regions prior to exiting. This causes mouse pointer regions to remain defined for subsequent applications. The problem is not exhibited unless a subsequent application enables the mouse pointer regions. To ensure that all regions are removed for your application, use the following lines of code before adding your first region:

```
int count; id_count;
```

```
for ( count = 1; count <= 3; count++ )  
    for ( id_count = 0; id_count <= 255; id_count++ )  
        ms_delete_region( ( char ) id_count );
```

Also, your application should delete any mouse pointer regions prior to exiting.

Example

```
int ReturnCode;  
char RegionId;  
  
/* delete a region */  
ReturnCode = ms_delete_region( RegionId );
```

Bug

Before **ms_enable_regions** is initially called the application must call **ms_delete_region** with the pointer region equal to zero, to remove all possibly existing regions.

DeskMate Technical Reference

Mouse Manager

ms_disable_driver ()

ms_disable_driver turns off the mouse pointer and resets the mouse driver, but is not removed from memory.

Input Parameters

None.

Return Value

<int>

CSR_ERROR if the mouse is not available.

Example

```
int ReturnCode;  
  
/* turn off the mouse pointer and reset the driver */  
ReturnCode = ms_disable_driver();
```

See Also

ms_enable_driver()

DeskMate Technical Reference

Mouse Manager

ms_disable_regions ()

ms_disable_regions disables all of the defined mouse pointer regions.

Input Parameters

None.

Return Value

<int>

CSR_ERROR if the mouse is not available.

Special Notes

The mouse pointer appearance changes to the default arrow pointer and all of the mouse pointer regions are ignored until **ms_enable_regions** is called.

Example

```
int ReturnCode;  
  
/* disable all mouse pointer regions */  
ReturnCode = ms_disable_regions();
```

See Also

ms_enable_regions()

DeskMate Technical Reference Mouse Manager

ms_draw_pointer ()

ms_draw_pointer turns on the mouse pointer.

Input Parameters

None.

Return Value

<int>

CSR_ERROR if the mouse is not available.

Special Notes

Once **ms_draw_pointer** has been called, the mouse driver will automatically update the mouse pointer as the pointing device is moved. All CSR video routines that modify the display will turn the mouse pointer off before modifying the display, and restore the mouse pointer state before returning to the application. This can cause the mouse pointer to "flicker" when several video routines are called in succession. To avoid this problem, turn the mouse pointer off (with **ms_erase_cursor**) before calling the sequence of video routines and turn it back on after completing the video routine calls, or for simplicity turn the pointer off after an **event_read** and back on before the next **event_read**.

Example

```
int ReturnCode;  
  
/* returned from several video calls, so draw mouse pointer */  
ReturnCode = ms_draw_pointer();
```

See Also

ms_erase_pointer()

DeskMate Technical Reference Mouse Manager

ms_enable_driver ()

ms_enable_driver initializes the mouse driver.

Input Parameters

None.

Return Value

<int>

NULL if the mouse was installed.

CSR_ERROR if the mouse was not installed.

Special Notes

ms_draw_pointer must be called to display the mouse pointer after the mouse driver has been initialized with **ms_enable_driver**.

Example

```
int ReturnCode;  
  
/* initialize mouse driver */  
ReturnCode = ms_enable_driver();
```

See Also

ms_draw_pointer()

DeskMate Technical Reference

Mouse Manager

ms_enable_regions ()

ms_enable_regions enables all of the defined mouse pointer regions.

Input Parameters

None.

Return Value

<int>

CSR_ERROR if the mouse is not available.

Special Notes

ms_enable_regions re-enables all of the mouse pointer regions that were disabled with a call to **ms_disable_regions**.

Some versions of certain 1988 DeskMate applications did not delete their mouse regions prior to exiting. This causes mouse pointer regions to remain defined for subsequent applications. The problem is not exhibited unless a subsequent application enables the mouse pointer regions. To ensure that all regions are removed for your application, use the following lines of code before adding your first region:

```
int count; id_count;

for ( count = 1; count <= 3; count++ )
    for ( id_count = 0; id_count <= 255; id_count++ )
        ms_delete_region( ( char ) id_count );
```

Also, your application should delete any mouse pointer regions prior to exiting.

Example

```
int ReturnCode;

/* enable all mouse pointer regions */
ReturnCode = ms_enable_regions();
```

See Also

ms_disable_regions()
ms_delete_region()

Bug

Before **ms_enable_regions** is initially called the application must call **ms_delete_region** with the pointer region equal to zero, to remove all possibly existing regions.

DeskMate Technical Reference

Mouse Manager

ms_erase_pointer ()

ms_erase_pointer turns off the mouse pointer.

Input Parameters

None.

Return Value

<int>

CSR_ERROR if the mouse is not available.

Special Notes

The Event Manager will continue to return mouse events after **ms_erase_pointer** is called if the mouse driver is enabled. To prevent mouse events from being reported by the Event Manager, the application must call **ms_disable_driver**.

Example

```
int ReturnCode;  
  
/* about to make several video calls so erase mouse pointer */  
ReturnCode = ms_erase_pointer();
```

See Also

ms_disable_driver()
ms_draw_pointer()

DeskMate Technical Reference

Mouse Manager

ms_get_status (pMOUSE_STAT)
MOUSE_STAT *pMOUSE_STAT;

ms_get_status returns the current status of the mouse.

Input Parameters

pMOUSE_STAT is a pointer to a **MOUSE_STAT** structure.

Return Value

<int>

CSR_ERROR if the mouse is not available.

Special Notes

ms_get_status fills the specified structure with the current status of the mouse.

Example

*Should be
MOUSE_STAT*

```
int    ReturnCode;
{MOUSE_STRUCT    Ms_Status_Struct;
MOUSE_STAT {MOUSE_STRUCT    *pMs_Status_Struct;

pMs_Status_Struct = &Ms_Status_Struct;

/* get the current mouse status in the structure Ms_Status_Struct */
ReturnCode = ms_get_status( pMs_Status_Struct );
```

See Also

MOUSE_STAT structure at the beginning of this section.

Print Managers

"Print Managers"

Table of Contents

General Description/Notes	20- 1
Structures/Defines	20- 3
Device Printing	
ptd_close	Closes the output device.....20- 6
ptd_draw_list	Adds a form to the page list.....20- 7
ptd_get_page	Gets the page setup and page mode.....20- 8
ptd_new_line	Adds a new (blank) line to the page list...20- 9
ptd_open	Prompts the user for a device to print to..20-10
ptd_preload	Loads the configuration printer driver.....20-12
ptd_print_far_page	Prints text strings from a far segment.....20-13
ptd_print_page	Prints the contents of the page list.....20-14
ptd_put_nchars	Adds a string to the page list.....20-15
ptd_put_nchars_x	ptd_put_nchars at a specified margin.....20-16
ptd_select	Selects the print device to be used.....20-17
ptd_set_page	Sets the page setup and page mode.....20-18
ptd_start_page	Initializes a new page.....20-19
Direct Printing	
prt_close	Closes the raw print device.....20-20
prt_get_printer	Gets printer driver information.....20-21
prt_open	Opens and initializes raw print device.....20-22
prt_put_char	Puts character to print device.....20-23
prt_put_nchars	Puts string to print device.....20-24
prt_put_tty	Puts non-filtered char to print device.....20-25

General Description/Notes

DeskMate supports two types of printing. Device printing (specified by beginning with **ptd**) allows printing to the three devices; printer, screen, and file. Direct printing (specified by beginning with **pri**) only prints to the printer but allows the application much more control in the printing procedure. The two different types are explained in more detail below.

Device Printing

This section outlines the procedure entailed in printing to one of the following logical devices:

- Printer
- Screen
- File

A dialog box will ask the user if they wish to:

- Print to the Screen
- Print to the Printer
- Print to a File

If the user chooses to print to the screen, the application must erase its menu bar. Print To Device (PTD) then creates a viewport that covers all but the bottom two rows of the physical screen, leaving these rows for the application to display buttons that may control the operation of the screen. PTD processes user input for cursor movement, and paging the display. PTD returns when it receives an outside event from **event_read** or the user chooses to quit.

PTD will use a buffer called the page list to hold pointers to the textual and graphic information which make up the current page. The first call to PTD should be **ptd_open** to initialize the page list for the current page and get the print device. **ptd_start_page** is called to begin each new page. Subsequent PTD calls build a page list for the current page. **ptd_print_page** is called to actually print the page. After printing is finished, it is the responsibility of the application to call **ptd_close** to close the print device, and to redraw its own menu bar and screen.

When allowing Print-To-Screen for a document that is greater than 160 characters, the Print-To-Screen user interface will only show the user the leftmost 80 characters and the rightmost 80 characters. If this situation is not appropriate for your needs, then your application should gray the screen option for those documents or use an alternate mechanism if the user selects screen as the print device.

Page List Format

The page list contains pointers to the textual and graphic components which make up the current page. The actual information to be displayed on the page remains within the application routine calling PTD.

Each entry of the page list points to either a line of text, or a display list for a picture to be displayed on the page. There will be one entry in the page list for every line of text and for every region of graphics, so the maximum number of entries is the maximum number of lines on a page.

Each page list entry consists of a pointer to the element, a byte to indicate the element type as text line or display list, and a byte to indicate the length if the element is a text string.

DeskMate Technical Reference

Print Managers

The display lists are in the format produced by the form manager. Coordinates within each display list are relative to the boundaries of the list, not relative to the position of the picture on the page.

The text lines each can contain a maximum of 132 printable ASCII characters along with embedded control characters. Control characters for underline on/off and boldface on/off are supported. See CSRVID.H for defines.

Direct Printing

The direct print subroutines allow a direct interface to the printer device LPT1, LPT2 or LPT3. The device can be opened or closed using these calls. Data may be sent to the printer through a filtered subroutine which allows character attributes to be changed, or directly to the printer. The `prt_get_printer` subroutine will allow access to the currently loaded printer.

DeskMate Technical Reference Print Managers

Structures/Defines

```
/*-----*/
/*
/* CSRPT.H contain the Print structures and defines */
/*
/*-----*/

/* Printer margin */
struct margin_defn
{
    char          left;          /* left margin */
    unsigned char lwidth;        /* printed line width */
    char          linepp;        /* total lines per page */
    char          plinepp;       /* printed lines per page */
};
typedef struct margin_defn MARGIN;

/* Page setup */
struct pgsetup_defn
{
    char          mode;          /* LANDSCAPE or PORTRAIT */
    MARGIN        mNotebk;       /* NOT SUPPORTED margin defs */
    MARGIN        mLandscp;      /* landscape margin defs */
    MARGIN        mPortrait;     /* portrait margin defs */
    char          bDspace;       /* double space boolean flag */
    char          bPgpause;      /* pause between pages flag */
    char          bScontrol;     /* send control sequences flag */
    char          bGraphic;      /* graphic mode boolean flag */
    char          bText;         /* text mode boolean flag */
};
typedef struct pgsetup_defn PGSETUP;

/* Page setup options */
struct pgopt_defn
{
    unsigned char max_line_wid;  /* max width */
    char          bMode;         /* mode enable flag */
    char          bLmargin;      /* left margin enable flag */
    char          bLwidth;       /* line width enable flag */
    char          bLinepp;       /* total lines/page enable flag */
    char          bPLinepp;      /* printed lines/page enable flag */
    char          bDspace;       /* set double space flag */
};
typedef struct pgopt_defn PGOPT;

/* Page modes */
struct pgmode_defn
{
    PGOPT         pNotebook;     /* NOT SUPPORTED pg setup opts */
    PGOPT         pLandscape;    /* landscape pg setup opts */
    PGOPT         pPortrait;     /* portrait pg setup opts */
};
typedef struct pgmode_defn PGMODE;

/* Printout types */
PORTRAIT        0
LANDSCAPE       1
```

DeskMate Technical Reference Print Managers

```

struct printer_defn
{
    char          bTextOnly;
    char          bIBMCompat;
    char          bCROnly;
    char          name[12];
    unsigned char max_columns; — 80 or 132
    unsigned char paper_sizes;
    unsigned char line_pitches;
    unsigned char char_pitches;
    unsigned char char_sets; } not used
    unsigned char font_styles; }
    unsigned char print_mode;
    unsigned char bFormFeed;
    unsigned char unaddr_width; } 0 or 5; don't know the units. Default unaddressable region is 1/4"
    unsigned char unaddr_height; } on each side. This is in addition.
    unsigned char orientations;
    unsigned char font_attrs;
    unsigned char nColors; — changed
    unsigned int  horz_dpi; } One of these: 150x150, 120x60, 60x60, or 0x0 (for text-only)
    unsigned int  vert_dpi; } 360x180
};
typedef struct printer_defn PRINTER;

PRT_TEXT_ONLY      SELECTED
PRT_GRAPHICS       DESELECTED

PRT_IBM            SELECTED
PRT_TANDY          DESELECTED

CR_ONLY            SELECTED
CR_WITH_LF         DESELECTED

PRT_A4             0x01
PRT_A5             0x02
PRT_B5             0x04
PRT_LEGAL          0x08
PRT_LETTER         0x10
PRT_HALF           0x20

PRT_3LPI           0x01
PRT_6LPI           0x02
PRT_8LPI           0x04

PRT_10CPI          0x01
PRT_12CPI          0x02
PRT_16CPI          0x04

PRT_CS_TANDY       0x01
PRT_CS_ASCII       0x02
PRT_CS_IBM1        0x04
PRT_CS_IBM2        0x08

PRT_STANDARD       0x01
PRT_CORRESPOND     0x02

PRT_UNIDIRECT      0x01
PRT_BIDIRECT       0x02
PRT_SHEET          0x04
PRT_CONTINUOUS     0x08

PRT_PORTRAIT       0x01
PRT_LANDSCAPE      0x02

```

DeskMate Technical Reference Print Managers

```
PRT_ELONGATED      0x01
PRT_DBL_HEIGHT     0x02
PRT_SUBSCRIPT      0x04
PRT_SUPERSCRIPT    0x08
PRT_BOLD           0x10
PRT_ITALIC         0x20 /* not supported */
PRT_UNDERLINE      0x40
PRT_OVERSCORE      0x80

/* Print-To-Device */
PTD_READY          0
PTD_CANCEL         CSR_ERROR
PTD_TO_SCREEN      0
PTD_TO_PRINTER     1
PTD_TO_FILE        2

/* Device flags for ptd_open */
PTD_DEVICES        0x07 /* all devices (screen, printer, file) */
PTD_SCREEN         0x01 /* mask for screen device */
PTD_PRINTER        0x02 /* mask for printer device */
PTD_FILE           0x04 /* mask for file device */

/* Character Y extent sizes with printer CPI scales */
PRT_10CPI_YEXT     166 /* 166.67 = yext of chars in a 10 cpi draw list */
PRT_12CPI_YEXT     200 /* 200.00 = yext of chars in a 12 cpi draw list */
PRT_16CPI_YEXT     277 /* 277.77 = yext of chars in a 16 cpi draw list */
```

DeskMate Technical Reference Print Managers

ptd_close ()

ptd_close closes the ptd output device.

Input Parameters

None.

Return Value

<int>

CSR_ERROR if an error occurred during the close.

NULL if the close was successful.

Special Notes

Do not call **ptd_close** if PTD_TO_SCREEN was selected.

DeskMate Technical Reference

Print Managers

ptd_draw_list (pFormHdr, Column, Cpi)
FORM_HDR *pFormHdr;
char Column;
char Cpi;

ptd_draw_list will add a pointer to the indicated form onto the current page list.

Input Parameters

pFORMHDR is a pointer to a form.

Column is the character column at which the form is to be displayed.

Cpi is the character define for the cpi setting of the printer.

Return Value

CSR_ERROR if the page list is full, the specified form is too small, or the page setup has not been initialized.

Special Notes

An entry pointing to the form will be appended to the page list with a bounding rectangle determined by the current line and the indicated column.

form_open must be called before calling **ptd_draw_list**.

The cpi (characters per inch) settings are:

PRT_10_CPI

PRT_12_CPI

PRT_16_CPI

ptd_draw_list will return CSR_ERROR if the specified form is too small or if the page setup has not been initialized with a call to **ptd_set_page**.

See Also

ptd_set_page()

DeskMate Technical Reference

Print Managers

ptd_get_page (pSetup, pMode)
PGSETUP *pSetup;
PGMODE *pMode;

ptd_get_page gets the current printer page setup and page mode.

Input Parameters

pSetup is a pointer to a PGSETUP structure where the current page setup will be stored.
pMode is a pointer to a PGMODE structure where the current page mode will be stored.

Return Value

None.

DeskMate Technical Reference Print Managers

ptd_new_line ()

ptd_new_line adds an entry for a blank line to the page list.

Input Parameters

None.

Return Value

CSR_ERROR if the page list is full.

DeskMate Technical Reference Print Managers

ptd_open (Devices)
int Devices;

ptd_open displays a dialog box prompting the user whether to print to the screen, the printer, or a file. Based on the response, it initializes relevant internal variables, and initializes the page list to subsequently build the current page.

Input Parameters

Devices specifies if any of the devices should be disabled for printing. The following defines are in `csrprt.h`:

<code>PTD_DEVICES</code>	all devices (screen, printer, file) are enabled.
<code>PTD_SCREEN</code>	screen device.
<code>PTD_FILE</code>	file device.
<code>PTD_PRINTER</code>	printer device.

Use a combination of the above defines to disable a particular device.

Return Value

<int>

`PTD_CANCEL` if the user chose to cancel the print.
`PTD_TO_SCREEN` if the user chose to print to the screen.
`PTD_TO_PRINTER` if the user chose to print to the printer.
`PTD_TO_FILE` if the user chose to print to a file.

Special Notes

ptd_open requires the application to make the `guf_bind_init` call.

ptd_open determines the device to send the printed output to and opens the print device. If the return value from this call is `PTD_CANCEL`, the application may continue without printing or making further printing calls. If the return value is `PTD_TO_SCREEN`, the application is responsible for erasing its menu bar before calling `ptd_print_page` for the first time. After calling **ptd_open**, regardless of the return value, it is the responsibility of the application to redraw its screen in order to erase the print-device dialog box that was displayed.

If print to screen is disabled, the application must check to see if there is a printer driver enabled. If there is not a printer driver, the application MUST gray print on the file menu (and anywhere else it appears.)

ptd_open grays print to file and print to printer if there is no printer driver.

Applications must also gray print if print to screen is disabled AND the printer is TEXT ONLY and the page setup mode is not PORTRAIT. (see example #2)

When allowing Print-To-Screen for a document that is greater than 160 characters, the Print-To-Screen user interface will only show the user the leftmost 80 characters and the rightmost 80 characters. If this situation is not appropriate for your needs, then your application should gray the screen option for those documents or use an alternate mechanism if the user selects screen as the print device.

DeskMate Technical Reference Print Managers

Example #1

```
/* allow printing to a file or a printer, */  
/* and screen printing is disabled. */  
ptd_open( PTD_DEVICES - PTD_SCREEN )
```

Example #2

```
ptd_get_page( &page_setup, &page_mode );  
prt_get_printer( &printer_struct );  
  
if ( printer_struct.bTextOnly == TRUE && page_setup.mode != PORTRAIT )  
    /* gray print */
```

See Also

```
cfg_get_prt_data()  
guf_bind_init()
```

DeskMate Technical Reference

Print Managers

ptd_preload ()

ptd_preload loads the printer driver specified by the printer configuration.

Input Parameters

None.

Return Value

CSR_ERROR if the printer driver was not loaded.

Special Notes

When printing data which exists exclusively on a removable disk, problems may occur if the printer driver and the source data file are on separate disks. The application should use **ptd_preload** to preload the printer driver, check for the source data file and then call **ptd_open**.

Bugs

If the user has selected None (or No printer) for the printer driver and this call is made the CSR will crash under 3.00, 3.01, and 3.02 DeskMate core systems. Under 3.03, 3.04 and 3.05 a message box will be displayed indicating that the printer driver on the disk is not a sufficient version for that version of DeskMate. The later is not harmful to the system, but is somewhat misleading to the user as the true problem is that there is no printer driver selected. To work around these problems an application should first call **cfg_get_prt_data** to determine if the user has a printer selected. If the printer is set to None (or No printer) the application should skip the **ptd_preload** call and continue as appropriate.

DeskMate Technical Reference

Print Managers

ptd_print_far_page (bLastPage, Segment)

char bLastPage;

int Segment;

ptd_print_far_page allows the caller to store the text strings to be printed in a segment other than his data segment. If **ptd_print_far_page** is called only text strings may be reference by the page list.

Input Parameters

bLastPage indicates if the page is the last to be printed.

Segment is the address of the segment in which the data resides.

Return Value

CSR_ERROR if the user selected cancel.

NULL if the print was successful.

DeskMate Technical Reference

Print Managers

ptd_print_page (bLastPage)
char bLastPage;

ptd_print_page sends the current contents of the page list to the print device.

Input Parameters

bLastPage should be set to TRUE if the page is the last to be printed.

Return Value

<int>

NULL if the print was successful and the application may proceed with further printing.

CSR_ERROR if the user chose CANCEL during the print, or an error occurred during the print. If this value is returned, the application must discontinue the print. If the application receives CSR_ERROR as the return value, the application must not call **ptd_close**

Special Notes

If the page being printed is the first page of the current print job, the printer output device will be opened before the print begins. **ptd_print_page** looks at the current page setup to determine the amount of the form to print. If an error occurs during the print, or the user cancels the print, **ptd_print_page** will automatically close the print device before returning to the application, and therefore, if the application receives CSR_ERROR as the return value, the application must not call **ptd_close**. If page setup specifies to print in character mode, any graphic elements in the form will be ignored, and only the text portion of the form will be printed. Any error conditions that may arise during the printing of the form are handled entirely within PTD, including displaying error messages and interacting with the user.

The application should never call **ptd_print_page** without first calling **mb_erase** when printing to screen.

DeskMate Technical Reference

Print Managers

ptd_put_nchars (pString, Length)

char *pString;

int Length;

ptd_put_nchars adds a printer indicated string to the page list at the current line.

Input Parameters

pString is a pointer to a string of ASCII characters.

Length specifies the number of characters in the string.

Return Value

CSR_ERROR if the page list is full.

Special Notes

ptd_put_nchars adds an entry to the page list which points to the indicated string. The string must not exceed 132 printable characters. The DeskMate text attributes will be honored for boldface and underline.

10h	UNDERLINE OFF
11h	UNDERLINE ON
12h	BOLD OFF
13h	BOLD ON

The string is not copied, only the pointer is stored, not the contents.

DeskMate Technical Reference Print Managers

ptd_put_nchars_x (pString, Length, Margin)

```
char *pString;  
int   Length;  
int   Margin;
```

ptd_put_nchars_x allows a text line to be printed to a device at a specified margin.

Input Parameters

pString is a pointer to an ASCII string.

Length is the number of characters in the string.

Margin is the character X position at which to print the string.

Return Value

CSR_ERROR if the page list is full.

DeskMate Technical Reference

Print Managers

ptd_select (DeviceType, FileHandle)

int DeviceType;
int FileHandle;

ptd_select selects the print device to be used. It also sends the startup commands when printing to the printer.

Input Parameters

DeviceType is a defined device type: PTD_TO_PRINTER, PTD_TO_SCREEN, or PTD_TO_FILE. **FileHandle** is an integer file handle and is only used if **DeviceType** is PTD_TO_FILE.

Return Value

CSR_ERROR if the print device cannot be selected.

Special Notes

ptd_open calls **ptd_select**, therefore the application should only make calls to one and not to both.

pri_open does *NOT* call **ptd_select**.

DeskMate Technical Reference Print Managers

ptd_set_page (pSetup, pMode)
PGSETUP *pSetup;
PGMODE *pMode;

ptd_set_page sets the current printer page setup and page mode.

Input Parameters

pSetup is a pointer to a PGSETUP structure which contains the desired page setup.

pMode is a pointer to a PGMODE structure which contains the desired page mode.

Return Value

None.

DeskMate Technical Reference Print Managers

ptd_start_page (CharAttrOverride)
char CharAttrOverride;

ptd_start_page initializes a new page.

Input Parameters

CharAttrOverride allows the caller to override the character attributes of the previous page.

Return Value

None.

Special Notes

CharAttrOverride may be set to any combination of NORMAL, BOLD, and UNDERLINE and should be set to CSR_ERROR to leave the current character attributes intact.

ptd_start_page uses the current page setup to determine its actions.

DeskMate Technical Reference Print Managers

prt_close ()

prt_close closes the raw printer device LPT1, LPT2 or LPT3.

Input Parameters

None.

Return Value

None.

DeskMate Technical Reference Print Managers

```
prt_get_printer ( pPRINTER )  
PRINTER *pPRINTER;
```

`prt_get_printer` gets information on the currently loaded printer driver.

Input Parameters

`pPRINTER` is a pointer to a `PRINTER` structure where the printer driver information will be stored.

Return Value

<int>

`CSR_ERROR` if "none" was selected in printer setup.

*This call can return CSR_ERROR even if it succeeds.
Ignore the return code.*

DeskMate Technical Reference Print Managers

prt_open ()

prt_open opens and initializes the raw printer device LPT1, LPT2 or LPT3 as specified by the printer setup accessory. **prt_open** does **NOT** send any start up commands to the printer.

Input Parameters

None.

Return Value

<int>

CSR_ERROR if the printer is unavailable.

Special Notes

prt_open does not call **ptd_select**. **ptd_select** calls **prt_open**. Typically an application should initialize "direct printing" by calling **ptd_select**. This will assure that the printer is set to the correct CPI, etc. It is especially important for laser printers since the start up commands will also override the laser top margin defaults of 1/2 inch and set the top margin to zero. In addition, total printed lines per page is changed from the default of 60, to 66.

This call can crash the machine under DM 3.02 or earlier if no printer driver is selected. See ptd_preload(), p. 20-12 — same thing.

DeskMate Technical Reference Print Managers

prt_put_char (Character)
char Character;

prt_put_char outputs the character or the attribute sequence indicated by **Character**.

Input Parameters

Character is an ASCII character value. If **Character** is a control code, it will be interpreted and the appropriate attribute sequence will be produced.

Return Value

CSR_ERROR if an error occurred.

Special Notes

If **Character** is in the range 20h to FFh it will be passed directly to the printer. If **Character** is one of the following, the appropriate sequence will be sent:

10h	UNDERLINE OFF
11h	UNDERLINE ON
12h	BOLD OFF
13h	BOLD ON

DeskMate Technical Reference

Print Managers

prt_put_nchars (pString, nChars)

char *pString;
int nChars;

prt_put_nchars prints a string directly to the printer device.

Input Parameters

pString is a pointer to a string.

nChars is the number of characters in the string.

Return Value

MSG_RETRY if an error occurred and the user wishes to RETRY.

MSG_CANCEL if an error occurred and the user wishes to QUIT.

Anything else means success.

Special Notes

The string to be printed may include embedded attributes. The **nChars** parameter indicates the number of character and attribute bytes in the string.

The MSG_RETRY and MSG_CANCEL codes are defined in CSRCMPS.H.

DeskMate Technical Reference

Print Managers

prt_put_tty (Character)
char Character;

prt_put_tty outputs **Character** directly to the printer.

Input Parameters

Character is any ASCII value.

Return Value

CSR_ERROR if an error occurred.

Special Notes

Sends the character directly to the printer with no filtering.

**Tandy Digital
Sound Library**

"Sound Library"

Table of Contents

Introduction and Overview.....	21- 1
Hardware	21- 1
Software	21- 3
Structures/Defines	21- 5
snd_addbuf	Adds memory for the player to use.....21- 9
snd_amplify	Generates function table to amplify sound.21-10
snd_apply_func	Applies a function table to a sound.....21-11
snd_compensate	Creates a recording translation table.....21-12
snd_compress_part ...	Performs data compression on a sound.....21-13
snd_cue	Puts hardware in standby for play/record..21-14
snd_decompress_part .	Performs data decompression on a sound....21-15
snd_exit	Turns the sound chip off.....21-16
snd_flush	Starts playing queued sounds.....21-17
snd_init	Initializes the sound chip.....21-18
snd_play	Plays sound(s) in memory.....21-20
snd_record	Records a sound into memory.....21-22
snd_stop	Stops the sound output or stops recording.21-24
snd_version	Returns the version number.....21-25
snd_volume	Sets the output level for play.....21-26
snd_wait	Waits for sound(s) to finish.....21-27
Appendix A	Compression/Decompression.....21-28
DeskMate 88	DeskMate Sound 1988 compression.....21-29
Appendix B	Bios Digital Sound Support.....21-33
Appendix C	DeskMate Sound 1988 File Format.....21-35
Appendix D	DeskMate Environment Issues.....21-37

DeskMate Technical Reference

Tandy Digital Sound Library

Introduction and Overview

The Tandy Digital Sound Library is for use with machines with built-in digital sound hardware. These machines come in two different series. The 1000 series which include the Tandy 1000 SL, Tandy 1000 TL, and the Tandy 1000 TL/2. These machines include not only the built-in digital sound hardware, but they also include the joystick hardware. A second series of machines are the non-joystick series. These machines have the built-in digital sound hardware, but do not include the complete joystick circuitry.

The digital sound hardware that is incorporated into these systems can be used to record audible sounds and store them in digital form. These recorded sounds can then be stored on disk, transmitted by modem, or manipulated and combined with other recorded sounds. These sounds can then be "played back" by converting the digital information back into audible sounds. The Sound Library is designed to hide the complexities and requirements of the digital sound hardware from the software developer. It provides a set of functions that can be used to record, play, and manipulate sounds.

The Hardware

The digital sound hardware is an integral part of the Tandy 1000 SL/TL and TL/2 joystick circuitry. The key component is an eight-bit digital-to-analog converter, also known as a DAC. This device converts an eight-bit number into a corresponding voltage. Each bit that is turned on produces a specific amount of voltage. For example, Bit 1 produces twice as much voltage as Bit 0, Bit 2 produces twice as much voltage as Bit 1, and so on. The sum of all 8 bits (on or off) becomes the voltage that is amplified and sent on to the speaker. By varying the voltage output at regular intervals, an audible sound can be produced.

The method used to quantize, or digitally record, an analog sound is called Successive Approximation. When an input signal is to be digitized, several events occur. First, the voltage on the input is latched internally. This is done to prevent any signal variation from affecting the approximation of this sample. Then the same DAC that is used to create a sound begins generating eight different voltage levels by turning on and off bits in the DAC circuitry. The hardware determines if the voltage being input is equal or greater than the voltage the DAC outputs when a given bit is on. The most significant bit (7) is checked first to see if its level is greater than or equal to the level that was latched. If so, Bit 7 remains on and will be on in the finished byte. Otherwise, Bit 7 is turned off. Now Bit 7 is left as it is and the same type of comparison is done on Bit 6 and then on the remaining bits. The procedure resembles a binary search, in that each bit that is examined eliminates half of the remaining possible input signal levels. When all eight bits have been computed, the CPU can read the eight bit value present in the DAC and store it as one sample of the digital recording.

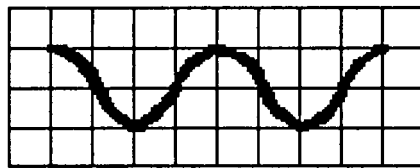
Since the same DAC is used by the joysticks and for both playing and recording, only one function can be done at a time. Therefore, the joysticks are disabled while the sound library is in use. (See `snd_init()` and `snd_exit()` in the reference section.)

To actually record or play back an audible sound, the record or playback hardware must obtain multiple samples by repeating these operations thousands of times a second. At the same time, software must manage the digitized data and get it to and from the hardware. Because it takes a minimum of two samples to reproduce a single cycle of an analog waveform, the highest frequency that can be recorded and reproduced is about one-half the sampling rate. This table can be used as a guide:

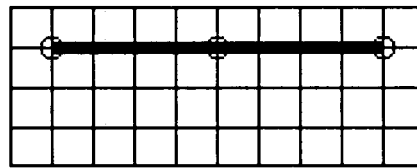
DeskMate Technical Reference Tandy Digital Sound Library

<u>Sampling Rate</u>	<u>Highest Recordable Frequency</u>
5,500 Hz	2,500 Hz
11,000 Hz	5,000 Hz
22,000 Hz	10,000 Hz

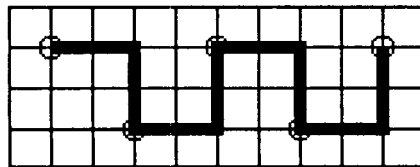
The higher the sampling rate, the higher the frequencies that can be recorded, while the lower frequencies will be recorded more accurately. For example, the following charts show a 5,500 Hz sine wave that is recorded at these three sampling rates and what the signal will look like in digitized form. Each sampling point is circled.



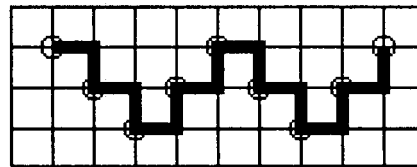
5.5 kHz Input Signal



Record / Playback at 5.5 kHz



Record / Playback at 11 kHz



Record / Playback at 22 kHz

If the sampling rate is too low, there is not enough information to produce something that sounds like the original sound. As the sampling rate increases the resulting waveform resembles the original more closely. A roll-off filter in the input hardware eliminates frequencies that could not be recorded even with the 22,000 Hz sampling rate. (A roll-off filter allows frequencies below a certain point to pass unchanged. Above that point, the higher the frequency, the less of the signal is allowed to pass.) Without this filter, significant distortion could be introduced into the recordings.

One specific form of distortion that arises in digital sound reproduction is called "aliasing". Most people have seen a form of aliasing when watching a movie where wheels that are in motion appear to turn backwards or stand still. The wheel appears to stand still because the wheel was turning at the same speed that the camera was exposing the film, and so the camera exposed each frame of the film when the wheel was in the same position. Aliasing in digital sound occurs when a frequency to be recorded is equal or greater than half the sampling rate. When this occurs, the resulting recording will not reflect the frequency of the input signal. For example, the chart above showing the result of the 5.5 kHz signal being sampled at 5.5 kHz shows no change in the recorded signal. So because each sampling occurred when the signal was at the same voltage level, the signal appeared to stand still. The input roll-off filter helps reduce this and most other forms of aliasing in spurious high-frequency signals, but the sampling rate must still be set high enough to record the highest fundamental frequency expected.

The output section also has a roll-off filter to reduce high frequency distortion created in the D to A process as well as switching noise caused by the processor.

DeskMate Technical Reference

Tandy Digital Sound Library

To improve the resulting reproduction of a digitized sound recorded at a sampling rate lower than 22 kHz, the software computes intermediate samples so that 22,000 samples per second are still produced. This is called *oversampling*. At 11,000 Hz, one sample must be computed for every "real" one. At 5,500 Hz, three intermediate samples are computed. The intermediate samples are computed by taking the average between "real" samples. These extra samples result in a form of curve-smoothing being performed by making the change in amplitude between samples smaller. This reduces distortion in the output circuitry which appears when frequencies approach the limit of that sampling rate.

The Software

The functions (calls) provided in the Tandy Sound Library allow the caller to record and play sounds without having to deal with most of the hardware-related activities. Some of the key functions included in the sound library are discussed briefly below, and are discussed in more detail in later sections.

All functions included in the sound library are written to expect Microsoft C-style calling rules and all functions should be called FAR. The library functions further assume that SS is equal to DS. Only the registers SI and DI are preserved across library calls. All functions return result codes in AX (short), unless otherwise specified. Functions returning a long do so with the most significant word in DX and the least significant word in AX. A few functions return far pointers with the segment in DX and the offset in AX. All of these return formats are compatible with Microsoft C.

Since the developer may choose to use a different language than C, the developer's program is responsible for allocating all memory required by the sound library functions. The program informs the sound library of the address of most of these memory areas by calling **snd_addbuf()**. If the language requires the memory to be released before terminating, the developer's code is also responsible for this task after calling **snd_exit()**.

The sound drivers within the sound library make extensive use of interrupts and one DMA channel to operate the sound hardware. (The interrupt and DMA channel that are to be used are specified in the **snd_init()** call.) Because the interrupts must be serviced frequently, the developer must ensure that interrupts do not remain off for more than 1 millisecond at a time while playing or recording sounds. The DMA channel that the sound hardware uses must not be used for any reason during the entire use of the sound library, which starts with a call to **snd_init()** and ends with a call to **snd_exit()**.

The **snd_play()** call is used to output sounds located in designated buffers. Repeated calls to **snd_play()** will cause the sounds to be played back to back, assuming that the subsequent calls are made frequently enough. The sound data provided by the caller may be expanded (depending on the sampling rate) and is placed in a series of fixed-size buffers. These buffers are placed on a queue which is handed to an internal function that programs the DMA to run continuously in the same fixed-size area. If the DMA were allowed to stop, the brief time it takes to program it to point it at the next section of the sound and start it would be audible to the listener as a "wow" or "flutter" in the sound. So the DMA runs continuously through the same buffer and the sound driver inserts subsequent portions of the sound into the buffer when interrupts and other status indicate that the DMA is transferring data in other parts of the buffer. When the DMA returns to a given point, new sound information will be present. (This "circular" DMA mode is also used in recording.)

To help synchronize sound output with other activity such as animation, **snd_wait()** allows the caller to determine when a specific sound, previously supplied to **snd_play()**, is about to start outputting. The caller can wait, then begin displaying animation or perform other tasks that need

DeskMate Technical Reference

Tandy Digital Sound Library

to occur at or about the same time. The **snd_flush()** call forces the player to queue any residual sound data and makes sure that the player has started. The amount of data accumulated before the player starts automatically is determined by the number of buffers provided to **snd_addbuf()** and a parameter to **snd_init()**. The **snd_wait()** function will also allow the developer to determine when the player has output all of the sounds that have been queued. The **snd_stop()** function can either terminate the sound output immediately or at a certain point. The output level can be set with **snd_volume()**, but cannot be adjusted while sound is being produced.

The **snd_record()** call causes the hardware to start digitizing sounds and stores the resulting data in an area the developer specifies in the call. The **snd_wait()** call will inform the developer when recording has finished, and the **snd_stop()** call can terminate recording at any point. A "bucket-brigade" buffering system can be implemented by making multiple calls to **snd_record()**. This allows the sound library to record into one buffer and switch to a second, while the caller is processing the first and preparing a third. This allows continuous or extended recording.

The **snd_amplify()** function is the first of a series of functions that can be used to manipulate sound data. With this call, the amplitude of a section of sound can be increased or reduced. The **snd_amplify()** call merely prepares the information needed for the manipulation. The developer calls **snd_apply_func()** to indicate what sound is to be worked on. In this way, several sounds or only portions of a given sound can be modified.

Finally, **snd_compress_part()** allows digitized sounds to be compressed so that they require less memory for storage. Some compression methods can be adjusted to be more aggressive at the cost of time and sound quality, but the compressed sound will take even less space. The compressed sounds are passed to **snd_decompress_part()**, which restores them to a playable form.

DeskMate Technical Reference Tandy Digital Sound Library

Structures/Defines

These type definitions are used throughout the sound library. Some are for convenience (such as the unsigned typedefs below) and some are for portability or ease of change (such as `saddr`).

```
typedef unsigned long   u_long;
typedef unsigned int    u_int;
typedef unsigned short  u_short;
typedef unsigned char   u_char;

typedef char far *      saddr;

typedef struct sound {
    saddr    buffer;      /* The sound buffer itself */
    u_long   sndlen;      /* Length of the sound in the buffer */
    u_long   buflen;      /* Length of the buffer itself */
    u_char   rate;        /* Rate the sound was recorded at */
    u_char   system;      /* Sound source (see defines below) */
    u_long   reserved[4]; /* Reserved for internal use */
} SOUND;
```

Utilized by:

`snd_record()`, **`snd_compress_part()`**, **`snd_apply_func()`**, and **`snd_decompress_part()`** and indirectly by **`snd_play()`**, **`snd_wait()`**, and **`snd_stop()`**.

The **SOUND** structure is used to describe a sound or a place to put a sound when recording. A **SOUND** structure is handed to **`snd_record()`**, and it fills in the elements of the structure which describe the sound. The caller sets the elements `buffer` and `buflen`, and **`snd_record()`** fills in the rest.

SOUND.buffer:

A pointer to the memory where the sound itself is stored.

SOUND.sndlen:

The length of the sound (in bytes). It should never be longer than `buflen`.

SOUND.buflen:

The length of the buffer in which the sound is stored. **`snd_record()`** uses `buflen`.

SOUND.rate:

Set by **`snd_record()`**. It is the sampling rate at which the sound was recorded. Use the following defines:

```
/* Sampling rate defines */
R5500   1      /* 5.5khz sample rate */
R11000  2      /* 11khz sample rate */
R22000  3      /* 22khz sample rate */
```

SOUND.system:

Set by **`snd_record()`**. It is the type of machine on which the sound was recorded. Use the following defines:

```
/* Machine sound came from */
B_SYNTH 0      /* Sound was synthetically produced */
B_MAC    0      /* Sound came from a Macintosh(R) */
B_AT8    1      /* Recorded on a 8mhz AT compatible */
B_SL     2      /* Recorded on a 1000-SL */
B_TL     3      /* Recorded on a 1000-TL */
```

DeskMate Technical Reference

Tandy Digital Sound Library

SOUND.reserved:

Reserved for internal use. Do **not** refer to or alter this field.

```
typedef struct sndhdr {
    /* The following items, except "ticket", "reserved1", and
       reserved2" are to be filled in by the caller */
    SOUND far *sndp;      /* The sound structure */
    u_long    start;      /* Where to start playing */
    u_long    end;        /* Where to stop playing */
    u_short   ticket;     /* When buffer was/is-to-be played */
    u_char    rate;       /* Rate sound is to be played at */
    u_char    reserved1;  /* Reserved for internal use */
    u_long    reserved2[4]; /* Also reserved for internal use */
} SNDHDR;
```

Utilized by:

snd_play(), **snd_wait()**, and **snd_stop()**.

A SNDHDR structure is used to describe to **snd_play()** what part of a sound to play, at what rate to play it, and where the sound is. A SNDHDR points to a SOUND structure. It is important not to confuse the two structures.

SNDHDR.sndp:

The pointer to the SOUND structure, which describes the sound to be played.

SNDHDR.start:

The index into the sound from which to start playing (the playing starts at SNDHDR.sndp->buffer[start]). A value of zero (0) means to play from the start of the sound.

SNDHDR.end:

The index into the sound before which the playing is to stop (the playing ends the byte *before* SNDHDR.sndp->buffer[end]). A value of zero (0) means play to the end of the sound.

SNDHDR.rate:

The rate at which the sound is to be played back. **snd_play()** ignores the rate field in the SOUND structure.

SNDHDR.ticket:

Used internally by the sound library to identify which play buffer contains the sound.

SNDHDR.reserved1:

SNDHDR.reserved2:

Elements which are reserved for internal use. Do **not** refer to or alter these fields.

DeskMate Technical Reference Tandy Digital Sound Library

```
typedef struct versioninfo {          /*Version format:
    char far * version;              /*Major[Minor](Edit)*/
                                     /*String containing version message
                                     that can be displayed via
                                     printf()*/
    u_char          major;           /*Major version number*/
    u_char          minor;           /*Minor version number 0=no
                                     character, 1=A, 2=B, 3=C...*/
    u_short         edit;            /*Edit number*/
    u_short         reserved1;        /*Reserved for future use*/
    u_long          reserved2;        /*Reserved for future use*/
    u_long          reserved3;        /*Reserved for future use*/
} VERSIONINFO;

/* Scratch area for internal use of compress/decompress routines */
typedef struct {
    u_long          scratch[25];      /* Scratch area */
} COMPINFO;
```

Utilized by:

snd_compress_part() and **snd_decompress_part()**. These routines store internal information about the compression or decompression in progress in this structure. Do *not* alter or refer to anything in this structure.

/* Defines for compress "type" parameter */
CTYPE_DESKMATE88 0

```
/* pParam structure for compressing with type=CTYPE_DESKMATE88 */
typedef struct compparam
{
    COMPINFO far *pinfo;             /* compress/decompress work area */
    u_long      start;               /* start position in sound */
    u_long      end;                 /* end position in sound(0 = to the end)*/
    u_short     compress_mode;        /* what kind of compression to do */

    /* following parameters used only by */
    /* compress_mode = DESKMATE88_ADJUSTABLE */

    u_short     threshold length;    /* minimum length of "silence" */
    u_char      threshold;           /* amplitude of "silence" */
    u_char      precision;           /* how accurate to make deltas */
} COMPPARAM;
```

COMPPARAM.pinfo:

Pointer to a COMPINFO structure. The COMPINFO structure need not be initialized to anything, it is a scratch area used by compress.

COMPPARAM.start:

The first sound sample to be included in the compression (the index to the first sample in the SOUND is zero (0).)

COMPPARAM.end:

The first sample to be *excluded* from the compression. That is, 1 plus the index of the last sample to be included in the compression. Zero (0) may be specified to indicate all bytes from **start** to the last byte of the sound.

DeskMate Technical Reference Tandy Digital Sound Library

COMPPARAM.compress_mode:

Indicates the parameters to be used in the compression. This field should be set to one of the following, depending on the sound source:

```
/* Defines for compparam "mode" parameter */
DESKMATE88 MUSIC          1      /* music compression */
DESKMATE88 SPEECH         2      /* speech compression */
DESKMATE88 ADJUSTABLE     3      /* adjustable compress mode*/
```

See Appendix A for more on compression and decompression.

```
/* Compression-precision default defines */
SND_PRECIS      1      /*"precision" compression default */
SND_SPEECH_THR  12     /*"threshold" speech comp. default*/
SND_MUSIC_THR   0      /*"threshold" music comp. default */
SND_THRESH_LEN  60     /* "threshold_length" default */

/* defines for snd_cue() "mode" parameter */
TOPLAY          1      /* prepare to play */
TORECORD        2      /* prepare to record */

/* defines for snd_init() "options" parameter */
FASTRAMP        0x0001 /* fast change between record and play */

/* Misc defines */
TRUE            1
FALSE           0

/* NULL */
NULL            0
```

These "defines" list all the error codes that are produced by calls in the sound library. Of the calls that do return error codes, most return at least INVALID and NOERROR. See the individual description of each call for specifics.

```
/* Error codes for sound calls */
BADFMT          -3     /* Decompress called with corrupted data or sound */
                  /* compressed with unsupported algorithm */
BADPARM          -2     /* Compress called with bad type or compress mode */
INVALID          -1     /* Call made when driver uninitialized or call not */
                  /* legal at this point */
NOERROR           0     /* Call succeeded */
INITBUF           1     /* Insufficient buffers were allocated before */
                  /* snd_init() was called */
INIT64K           2     /* All buffers spanned 64k addresses */
BADDMA           3     /* The DMA Channel requested is invalid */
BADIRQ           4     /* The IRQ number requested is invalid */
NOIRON           5     /* Can't find the sound hardware */
BADSIZE          6     /* Record buffer is too small or too small for the */
                  /* requested time */
WOULDBLOCK        7     /* snd_record() failed because two buffers are */
                  /* already queued and block was FALSE */
```

DeskMate Technical Reference Tandy Digital Sound Library

snd_addbuf (memory, count)
saddr memory;
u_short count;

snd_addbuf() assigns the memory pointed to by **memory** to the player's buffer pool.

Input Parameters

memory is a far pointer to the memory to be added to the player's buffer pool. If **memory** is NULL **snd_addbuf()** will return the size (in bytes) of a buffer.

count should be set to the number of buffers that will fit in the memory region pointed to by **memory**.

Return Value

NOERROR if successful.

INVALID if it is not valid to call **snd_addbuf()** at this point (i.e., **snd_init()** has been called or 128 buffers have already been added.)

Special Notes

snd_addbuf() may be called as often as desired to add buffers, until the maximum of 128 buffers have been added, or **snd_init()** is called. Any further calls to **snd_addbuf()** to add buffers will return INVALID.

To make use of the synchronizing capabilities of **snd_wait()**, the amount of buffer storage available must be greater than the size of the largest sound that could be passed to **snd_play()**. If the sounds to be played were recorded at 11 kHz, the amount of buffer space must be at least twice the size of the sound. For 5.5 kHz, at least four times the space is required.

The sound driver reserves three buffers for its internal use, and a minimum of two buffers are required for playing of sounds. Therefore, at least five buffer's worth of memory should be supplied before calling **snd_init()**. The three internal buffers are required even if **snd_play()** is not used. Also, at least one of the buffers supplied must be wholly contained in a physical 64K region (see the **snd_init()** description). This buffer is used to hold data that the DMA will read or write. The other two reserved buffers are used in certain functions for temporary storage. A minimum of two buffers (in addition to the three reserved buffers) are required to sustain continuous output.

See Also

snd_init()
snd_play()
snd_wait()
snd_exit()

DeskMate Technical Reference Tandy Digital Sound Library

```
void far snd_amplify ( factor, output )  
u_short factor;  
u_char far *output;
```

snd_amplify() generates a function table to be used to amplify (or de-amplify) a sound by the specified **factor**.

Input Parameters

output is a far pointer where the table will be placed and should be defined as a 256-byte array of type `u_char`.

factor specifies the level of amplification, which is multiplied by 1000. Values for **factor** greater than 1000 amplify and values below 1000 deamplify. (For example to amplify by 1.5 times, specify 1500.)

Return Values

None.

Special Notes

The table is then used in a call to **snd_apply_func()** to actually perform the amplification.

See Also

snd_apply_func()

DeskMate Technical Reference Tandy Digital Sound Library

```
void far snd_apply_func ( pSound, start, end, table )  
SOUND far *pSound;  
u_long start;  
u_long end;  
u_char far *table;
```

snd_apply_func() applies the function represented by **table** to the portion of the sound pointed to by **pSound** between **start** and **end**.

Input Parameters

pSound is a far pointer to a **SOUND** structure.

start is where to begin manipulating the sound.

end is where to stop manipulating the sound.

table is a far pointer to a 256-byte array of type **u_char**, obtained from a sound modification table generation routine, such as **snd_amplify()**.

Return Values

None.

See Also

snd_amplify()
snd_compensate()

DeskMate Technical Reference Tandy Digital Sound Library

snd_compensate (adj_table)
u_char far *adj_table;

snd_compensate() generates a translation table which can be used by **snd_record()** to adjust the recorded digital data to be centered around the digital centerline (0x80).

Due to parts tolerances in the analog sound hardware (used in recording just before digitization), the digital value of absolute silence can be different from one machine to another. It can also vary over long periods of time on a single machine.

Performing this compensation is not essential, but not using compensation can cause a "click" to be output at the beginning and the end of a sound, regardless of what machine it is played on. In addition, concatenating different sounds or combining different recordings may result in distortion if the recordings were not compensated.

Input Parameters

adj_table is a far pointer to a 256-byte table which will be filled with information on whether the analog centerline is above or below the correct position. This table is then supplied to **snd_record()**, which will adjust each sample as it arrives.

Return Values

NOERROR if successful.

INVALID if it is not valid to call **snd_compensate()** at this point (i.e., **snd_init()** has not been called.)

Special Notes

To compute the centerline **snd_compensate()** reads a few thousand samples which are then averaged. While the **snd_compensate()** call is being performed, there must not be any strong sound input from the microphone or line level input. If a microphone is in use, either unplug it or get the room quiet during this time. If a line level input is present, it should be disconnected or muted during the call. If an audio signal is present while computing the centerline, the result will probably not be as accurate as it could be because most sound waveforms are not symmetrical.

snd_compensate() need only be called before the first use of record. The resulting compensation table should be correct for several hours.

See Also

snd_record()
snd_apply_func()

DeskMate Technical Reference Tandy Digital Sound Library

long far **snd_compress_part** (pSound, dest, dest_size, type, pParam)
SOUND far *pSound;
u_char far *dest;
u_short dest_size;
u_short type;
COMPPARAM far *pParam;

snd_compress_part() performs data compression on the sound specified by **pSound**. **snd_compress_part()** is called repeatedly until the entire sound has been compressed.

Input Parameters

pSound is a far pointer to a SOUND structure.

dest is a far pointer to a buffer of size **dest_size** characters.

type specifies the algorithm to be used to do the compression. Currently there is only one **type**, CTYPE_DESKMATE88.

pParam is a far pointer to a COMPPARAM structure where additional compression details are specified.

Return Values

Greater than zero (0) if the data compression of this segment successful. (Return value is the length of the compressed data stored into the buffer pointed to by **dest**.)

Zero (0) if the data compression of entire sound is complete.

BADPARM if **type** is an unsupported compression algorithm

Special Notes

For each call to **snd_compress_part()** the buffer pointed to by **dest** is filled with up to **dest_size** characters, and then the number of characters stored there is returned. All calls but the first should specify NULL for **pSound**. On the last call, **snd_compress_part()** will return zero (0), meaning no bytes were stored in **dest**; when this happens, the compression is finished.

The contents of the **pParam** structure vary according to what **type** is set to. See Appendix A for a list of each **type** and the layout it uses for COMPPARAM.

See Also

Appendix A

DeskMate Technical Reference

Tandy Digital Sound Library

snd_cue (mode)
u_short mode;

snd_cue() puts the sound driver into standby mode for recording or playing.

Input Parameters

If **mode** is equal to TORECORD the driver is put into standby mode for recording.

If **mode** is equal to TOPLAY, the driver is put into standby mode for playing.

Return Values

NOERROR if successful.

INVALID the library is already recording or playing, or **snd_init()** has not been called.

Special Notes

When switching from play to record mode, or from record to play mode, a significant delay is encountered while the hardware is switched between modes (unless the FASTRAMP option has been specified to the **snd_init()** call). The length of the delay is dependent on the hardware in a given system, and can be over one second. In certain applications it can make cueing the start of a recording or synchronizing playback to video displays difficult, or impossible.

For example, a user could not be accurately alerted as to when recording would actually start if the delay is performed when **snd_record()** is initially called. By making the call **snd_cue(TORECORD)** before recording begins, it will perform the switch-over of the hardware, which will guarantee that recording will start immediately after calling **snd_record()**.

snd_cue() should only be called when switching from one mode (play or record) to the other. If called while the recorder or player is already running, or while the system is uninitialized, it will return an error.

A call to **snd_cue()** is not required in order to play or record. The hardware will be switched when **snd_record()** or **snd_play()** is called, if this has not already been done. However, there may be a delay before recording or playback begins.

snd_cue() does not need to be made if **snd_init()** was called with the FASTRAMP option.

See Also

snd_init()
snd_play()
snd_record()

DeskMate Technical Reference

Tandy Digital Sound Library

snd_decompress_part (pSound, pInfo, src, src_len)

SOUND far *pSound;

COMPINFO far *pInfo;

u_char far *src;

u_short src_len;

snd_decompress_part() decompresses a previously compressed sound. **snd_decompress_part()** is called repeatedly until the entire sound has been decompressed.

Input Parameters

pSound is a far pointer to a SOUND structure.

pInfo is a far pointer to a COMPINFO structure.

src is a far pointer to a buffer of **src_len** length.

Return Values

NOERROR if successful.

BADFMT if the compressed sound was compressed using an unsupported compression algorithm, or is corrupted

Special Notes

On each call, the buffer pointed to by **src** must contain **src_len** characters; these will be decompressed and stored in the specified sound. On all but the first call, NULL should be specified for **pSound**. A last call with **src_len** set to zero (0) should be made after all data has been passed through **snd_decompress_part()**.

Before calling **snd_decompress_part()**, the caller must ensure that the buffer associated with **pSound** is at least two bytes longer than the original uncompressed sound.

The caller must fill in the length of the *uncompressed* sound in the **pSound->sndlen** field, using the length of the sound before it was compressed. The original rate and bias values from the SOUND structure at the time of compression should be placed in the new SOUND structure before playing it.

DeskMate Technical Reference

Tandy Digital Sound Library

snd_exit (Options)

u_shourt Options;

snd_exit() calls **snd_stop()** if the player or recorder is running, disables the sound chip, and re-enables the joysticks. **snd_exit()** also restores the interrupt vector for the channel passed to **snd_init()** to the one that was present before **snd_init()** was called.

Input Parameters

Options is reserved for use in future releases. It should be set to zero for compatibility.

Return Values

NOERROR if successful.

INVALID if it is not valid to call **snd_exit()** at this point (i.e., **snd_init()** has not been called.)

Special Notes

A single call to **snd_exit()** does *not* release the memory previously supplied to **snd_addbuf()**. If a subsequent call is made to **snd_init()**, the sound library will attempt to re-use any memory that has been supplied to **snd_addbuf()** up to that point. Repeated contiguous calls to **snd_exit()** are not supported and may result in unexpected action in future releases.

See Also

snd_addbuf()

snd_init()

snd_stop()

DeskMate Technical Reference

Tandy Digital Sound Library

snd_flush ()

snd_flush() informs the player that there are no more sounds queued to be played in this group, and it is to immediately start playing what has been queued so far.

Input Parameters

None.

Return Values

NOERROR if successful.

INVALID if it is not valid to call **snd_flush()** at this point (i.e., **snd_play()** has not been called.)

Special Notes

If the quantity of sound queued had already reached the **trigger** threshold (see the **snd_init()** description), the player may already be running. Subsequent calls to **snd_play()** are perfectly valid, but the sound queued immediately after the **snd_flush()** will be preceded by a period of silence. The **snd_flush()** call is used to force the player to start after sounds have been queued, but before the quantity has reached the **trigger** threshold.

If the player is already running, any remaining space in the last buffer (up to 100 msec.) is filled with silence, then it is scheduled for playing. Failing to call **snd_flush()** or **snd_wait()** with the **wait** option TRUE after the last sound has been passed to **snd_play()** will result in up to 100 msec. from the end of the combined sound not being output.

When **snd_flush()** is called, any additional sound that is supplied to **snd_play()** will be separated from the earlier sounds by a period of silence. This is due to the fact that the sound driver (internal to the library) must have an exact amount of sound information in a buffer before it can be queued for playing. Calling **snd_flush()** gives the sound library permission to pad an incomplete buffer with silence and schedule it for playing. If the player runs out of sounds to play before a subsequent call to **snd_play()** can be processed, the player will go idle. The play queue will have to be filled to the trigger threshold or a call to **snd_flush()** or **snd_wait()** will be required to start the player again.

See Also

snd_init()
snd_play()
snd_wait()

DeskMate Technical Reference Tandy Digital Sound Library

snd_init (trigger, dma_ch, irq, options)

u_short trigger;
u_short dma_ch;
u_short irq;
u_short options;

snd_init() Initializes the sound chip, disables the joysticks, sets options, and prepares to play or record.

Input Parameters

trigger is used to indicate the number of buffers allowed to accumulate before the player is forced to start. A call to **snd_flush()** or to **snd_wait()** with the **wait** option set to TRUE will force the player to start before reaching the limit set by **trigger**. Setting **trigger** to zero (0) causes the player to fill all the supplied buffers before starting. A minimum of two buffers must accumulate before the player starts. If the **trigger** value is too low or there are not enough buffers to sustain continuous playing, the resulting sound will be broken or chopped. If this occurs, increase the number of buffers and/or set **trigger** to a higher value (or to zero).

dma_ch tells the library which DMA channel to use. Supplying a zero (0) will default to DMA channel 1. Due to various hardware restrictions, only DMA channels 1 and 3 can be used for sound. On the Tandy 1000 SL and 1000 TL and TL/2 machines, channel 1 must be used.

irq indicates which interrupt channel the sound hardware is set for. The 1000 SL/TL and TL/2 machines use interrupt channel 7. Supplying a zero (0) will default to interrupt channel 7.

options allows presetting various functions in the sound library. Once set, the **options** will remain in effect until **snd_exit()** is called. They are bit fields. All undefined bits are reserved for future use and should be set to zero.

FASTRAMP	Causes the switch to record from play, and from play to record to be performed instantly. When this option is selected, a "click" will be produced at the output port each time operating modes are changed. When this bit is set to zero (0), the change between operating modes is done as quietly as possible, but at the expense of time.
----------	---

Return Values

NOERROR if successful.

INITBUF if insufficient buffers were added before **snd_init()** was called.

INIT64K if all buffers spanned 64k addresses.

BADDMA if the DMA channel requested is invalid.

BADIRQ if the irq number requested is invalid.

NOIRON if the sound hardware was not found.

Special Notes

A **trigger** value that is greater than the number of available buffers is treated as zero, ie, the player will wait until all buffers are full before starting, unless **snd_wait()** or **snd_flush()** are called first. The number of available buffers at any instant can be up to three less than the number that were supplied to **snd_addbuf()**.

There is no check to confirm that the correct DMA and interrupt channel have been specified. Incorrect values will cause the sound library to malfunction.

DeskMate Technical Reference

Tandy Digital Sound Library

Once **snd_init()** is called successfully, any interrupts received on the specified channel are checked to see if they were generated by the sound hardware. If this is not the case, control is transferred to the routine that previously handled this interrupt. If this is a transient function, it must not delete the interrupt vector, mask the interrupt, or remove itself from memory without first calling **snd_exit()**.

See Also

snd_addbuf()
snd_cue()
snd_exit()
snd_flush()
snd_play()
snd_record()
snd_wait()

DeskMate Technical Reference Tandy Digital Sound Library

snd_play (pSndHdr)
SNDHDR far *pSndHdr;

snd_play() queues the sound specified by **pSndHdr** to be played.

Input Parameters

pSndHdr is a far pointer to a SNDHDR structure.

Return Values

NOERROR if successful.

INVALID if it is not valid to call **snd_play()** at this point (i.e., **snd_init()** has not been called.)

Special Notes

The sound is expanded and stored in the buffers supplied to **snd_addbuf()**. The sound will not begin to play until the number of buffers accumulated thus far is equal to or greater than the **trigger** parameter supplied to **snd_init()**. **snd_play()** returns immediately unless there are no more buffers available to queue the sound in. If no buffers are available and **snd_play()** has not finished expanding the specified sound, the player is started regardless of the setting of **trigger**.

snd_play() can be called repeatedly to output a series of sounds contiguously. Sounds recorded at different sampling rates can be played together as the differences in the playback rates are handled automatically by the player.

Once the player has started, it is up to the caller to make any additional calls to **snd_play()** at a rate that is greater than the rate the previously supplied sounds are being output at to keep continuous sound output. If the caller cannot keep ahead of the player, periods of silence will be inserted between sounds, each of which should not exceed 200 msec. The amount of silence depends on the given situation; but could be infinite if the application fails to make a call to **snd_wait()**, **snd_flush()**, **snd_stop()**, or **snd_exit()** before accepting keyboard input or performing a lengthy non-sound related activity.

If the recorder is active when **snd_play()** is called, the recorder will finish filling the specified buffers before **snd_play()** processes a request. To stop the recorder immediately, call **snd_stop()**.

After all sounds in a group have been passed to **snd_play()**, a call to **snd_flush()** or to **snd_wait()** with the **wait** option equal to TRUE must be made, or a maximum of 100 msec. of the end of the combined sound provided to **snd_play()** will not be output. It will be played at the beginning of the next sound passed to **snd_play()**.

DeskMate Technical Reference Tandy Digital Sound Library

See Also

snd_addbuf()
snd_cue()
snd_exit()
snd_flush()
snd_init()
snd_stop()
snd_wait()

DeskMate Technical Reference

Tandy Digital Sound Library

```
snd_record ( pSound, maxTime, rate, adj_table, block )  
SOUND far *pSound;  
u_short maxTime, rate;  
u_char far *adj_table;  
u_short block;
```

snd_record() places the hardware into record mode at the sampling rate specified by **rate** and fills the buffer described by **pSound** until **maxTime** milliseconds have passed or the buffer is full.

Input Parameters

pSound is a far pointer to a SOUND structure.

maxTime is the number in milliseconds for which the sound is to be recorded. If **maxTime** is zero (0) the buffer will be filled to the size specified by **buflen** in the SOUND structure. When it is non-zero, the value is converted to a number of samples to record. **buflen** is then checked to see if that number of bytes are available. If not, an error is returned.

The number of bytes per millisecond for each sampling rate is as follows:

<u>Sampling Rate</u>	<u>Bytes per millisecond</u>
22 kHz	22
11 kHz	11
5.5 kHz	5.5

Note that **snd_record()** will round **maxTime** up to the nearest number of *even* milliseconds. So, if **maxTime** is odd, and the buffer is unable to hold the extra bytes required for the additional millisecond of recording time, **snd_record()** will return **BADSIZE**.

adj_table is a far pointer to a translation table that will be used on each sample that is recorded. It is identical to the table filled in by such functions as **snd_compensate()**. The table must be filled in by calling **snd_compensate()** before recording begins. If this translation is not desired, call **snd_record()** with **adj_table** equal to "(u_char far *)NULL".

Once the recording process begins, **snd_record()** returns so that additional buffers can be prepared and **snd_record()** can be called again. If a second call to **snd_record()** is made before the first buffer has been filled, **snd_record()** will return, and the second buffer will be used after the first one is filled. If a third call is made before the first buffer is filled, **snd_record()** will wait until the recorder starts using the second buffer before returning, or it will return immediately with an error, depending on the **block** parameter (see below). If there is always an extra buffer available to the recorder (except at the end of recording), the group of buffers represent a continuous recording. The sampling rate specified in the first of a series of calls to **snd_record()** will be the rate used throughout the recording.

Recording stops when **snd_record()** has filled the buffers supplied in one or more calls, **snd_stop()** or **snd_exit()** is called, or **maxTime** is reached. After recording stops, the next call to **snd_record()** will set a new sampling rate.

block allows the user to choose to wait until the buffer that is specified in the call can be queued, or to return immediately. (This is useful only when using more than two buffers for recording. The first two calls always return immediately.) If **block** is set to **TRUE**, **snd_record()** will wait until the active buffer is filled. Then, **snd_record()** will return. If **block** is **FALSE** and **snd_record()** already has a standby buffer queued (ie, two calls to **snd_record()** have already been made), it will return immediately with an error. The calling program can then scan the keyboard or perform other actions, and then call **snd_record()** with the same buffer. This can be done repeatedly until the buffer is accepted.

DeskMate Technical Reference

Tandy Digital Sound Library

Return Values

NOERROR if successful.

INVALID if it is not valid to call **snd_record()** at this point (i.e., **snd_init()** has not been called.)

BADSIZE if the buffer was smaller than 100 bytes or it was too small to hold the requested time.

WOULDBLOCK if two buffers are already queued and **block** was set to FALSE.

Special Notes

Recording buffers must be at least 100 bytes long, although less than that amount can be used by setting **maxTime** appropriately.

As each buffer is used, the recording rate and bias fields in each sound structure are updated. This information should never be changed or discarded, otherwise the sound may play back at the wrong speed or pitch.

Addendum

The **adj_table** parameter to **snd_record()** is ignored in this release of the Tandy Digital Sound Library. All subsequent releases of the Sound Library will honor this parameter.

In the meantime, you can achieve the same results by calling **snd_compensate()** to generate the table, perform the recordings with one or more calls to **snd_record()**, and then call **snd_apply_func()** to adjust the recorded sounds. A call to **snd_apply_func()** must be made for each buffer filled by **snd_record()**.

If the workaround described above is used, **snd_record()** should be called with the **adj_table** parameter equal to (char far *)NULL. By doing this, the code will continue to function as intended with subsequent releases of the Sound Library, when the **adj_table** parameter will be honored. Otherwise, the adjustment will be performed twice.

See Also

snd_compensate()

snd_exit()

snd_init()

snd_stop()

DeskMate Technical Reference

Tandy Digital Sound Library

```
void far snd_stop ( pSndHdr )  
SNDHDR far *pSndHdr;
```

snd_stop stops the sound output or stops recording.

Input Parameters

pSndHdr is a far pointer to a SNDHDR structure.

Return Values

None.

Special Notes

While in play mode, if **pSndHdr** is NULL and the player is running, output stops within 100 msec. If **pSndHdr** is NULL and the player is not running, no sound will be output. If **pSndHdr** is not NULL, the player is not running, and the first sound in the queue is the one that is to be the stopping point, no sound is output.

Otherwise, when **pSndHdr** is not NULL, **snd_flush()** is called internally, sound output continues until the specified sound is encountered, and then output stops. If the sound is not in the play queue (it has already played or was never in the queue), output stops within 100 msec. as when **pSndHdr** is NULL. If **snd_flush()** is not called before **snd_stop()** is called and **pSndHdr** points to a sound that is within the last 100 msec. of sound passed to **snd_play()**, **snd_stop()** may not find it. If not, it will proceed as though **pSndHdr** were NULL.

After the player stops, all remaining sounds are removed from the play queue. **snd_stop()** then returns.

While in record mode, the **pSndHdr** parameter is ignored. Recording stops immediately. Up to 100 msec. of the most recently recorded sound will be discarded. The length of the sound stored in the record buffer is set, and if an additional buffer, provided by another call to **snd_record()**, was standing by, the length field in that buffer is set to zero (meaning, no contents).

See Also

snd_flush()
snd_play()

DeskMate Technical Reference

Tandy Digital Sound Library

VERSIONINFO far * far **snd_version** ()

snd_version() allows an application using the Tandy Digital Sound Library to determine which version of the library is being used, and to what features or options are available.

Input Parameters

None.

Return Values

A far pointer to the VERSIONINFO structure:

```
typedef struct versioninfo
{
    char far    *version;          /* String containing version
                                   message that can be displayed
                                   via printf() */
    u_char      major;             /* Major version number */
    u_char      minor;            /* Minor version number
                                   0 = no character, 1 = A,
                                   2 = B, 3 = C... */
    u_short     edit;              /* Edit number */
    u_short     reserved1;         /* Reserved for future use */
    u_long      reserved2;         /* Reserved for future use */
    u_long      reserved3;         /* Reserved for future use */
} VERSIONINFO;
```

Special Notes

When writing code that determines if an option is available, use the major and minor version fields. The edit number should not be used for this purpose.

snd_version() may be called at any time, even prior to calling **snd_init()**.

DeskMate Technical Reference

Tandy Digital Sound Library

```
void far snd_volume ( volume )  
u_short volume;
```

snd_volume() sets the hardware gain level to **volume** for the next time the player is started.

Input Parameters

volume must be a value in the range 1 (minimum output) to 7 (maximum gain). If the player is running, the previous value for **volume** will remain in effect until the player stops.

Return Values

None.

Special Notes

The state of the player can be checked by making the call **snd_wait((SNDHDR far *)NULL,0)**. If the result is zero (0), the player is inactive, and a new gain level will be set on the next call to **snd_play()**. Calling **snd_stop()** or **snd_record()** will also guarantee that the next sound output will be at the new gain level.

The gain level defaults to 7.

snd_volume() does not affect the sound data as **snd_apply_func()** does. It merely sets the hardware output level.

See Also

snd_apply_func()
snd_play()
snd_record()
snd_stop()
snd_wait()

DeskMate Technical Reference

Tandy Digital Sound Library

snd_wait (pSndHdr, wait)
SNDHDR far *pSndHdr;
u_short wait;

snd_wait() provides status on the player, recorder, or a specific sound in the play queue (possibly waiting for the specified sound, the player, or the recorder to finish) and then returns.

If **pSndHdr** points to a play structure and **wait** is equal to **TRUE**, **snd_wait()** will wait until the beginning of that sound is within approximately 100 msec. of being played, and then return **TRUE**. If the sound is not on the play queue (perhaps it has already started playing), **snd_wait()** returns **TRUE** immediately.

If **pSndHdr** is **NULL** and **wait** is equal to **TRUE**, **snd_wait()** waits for *all* sounds to be completely played or all recording buffers to be filled and then returns **TRUE**.

If **pSndHdr** is a pointer to a play structure and **wait** is equal to **FALSE**, **snd_wait()** will return **TRUE** if the beginning of that sound is within approximately 100 msec. of being played, it has already started playing, or it has been played already. It will return **FALSE** otherwise.

If **pSndHdr** is **NULL** and **wait** is equal to **FALSE**, **snd_wait()** will return **TRUE** if the player or recorder is running, and **FALSE** if not.

Input Parameters

pSndHdr is a far pointer to a **SNDHDR** structure or **NULL**. See above.

wait specifies when **snd_wait()** will return. See above.

Return Values

See above.

Special Notes

The **pSndHdr** parameter is ignored while the driver is in the record mode.

If the caller queues a single (or the first) sound and then calls **snd_wait()** with **pSndHdr** pointing to the first sound and **wait** equal to **TRUE**, **snd_wait()** will call **snd_flush()** (starting the player) and return immediately since the player is always started when **wait** is **TRUE**. The second and subsequent sounds on the queue can be waited on, provided sufficient sound buffers were provided via **snd_addbuf()**. See the **snd_addbuf()** description for information on computing the appropriate number of buffers.

See Also

snd_flush()
snd_addbuf()

DeskMate Technical Reference Tandy Digital Sound Library

Appendix A COMPRESSION/DECOMPRESSION

Compression is used to allow sounds to be stored in less space than they would normally require. Depending on the compression method used, significant space savings can be achieved, although some do so at the expense of sound quality.

When **snd_compress_part()** is called, the caller specifies which compression algorithm is to be applied, along with any other parameters the selected algorithm requires. The decompression call, **snd_decompress_part()**, determines which compression algorithm was used on the data and then applies the correct decompression algorithm.

The **COMPPARAM** structure is what is used to give a compression algorithm any parameters or options it needs. Thus its contents vary according to which compression algorithm is selected.

The remainder of this appendix discusses the compression types available and how to manipulate their options. The following compression algorithms are currently defined:

DESKMATE88 - Produces results identical to the compression used in 1988 DeskMate 03.02.0x Sound Editor and the 1988 DeskMate 03.02.0x Music application.

DeskMate Technical Reference

Tandy Digital Sound Library

DESKMATE 88

This compression and decompression algorithm produces results identical to the compression used in the DeskMate Sound Editor 1988 and the DeskMate Music 1988 application. To use this algorithm, the **type** parameter to **snd_compress_part()** needs to be **CTYPE_DESKMATE88** and the following fields in the **COMPPARAM** structure are filled in by the caller as follows:

COMPPARAM.pinfo:

Pointer to a **COMPINFO** structure. The **COMPINFO** structure need not be initialized to anything, it is a scratch area used by compress.

COMPPARAM.start:

The first sound sample to be included in the compression (the index to the first sample in the **SOUND** is zero (0).)

COMPPARAM.end:

The first sample to be *excluded* from the compression. That is, 1 plus the index of the last sample to be included in the compression. Zero (0) may be specified to indicate all bytes from start to the last byte of the sound.

COMPPARAM.compress_mode:

Indicates the parameters to be used in the compression. This field should be set to one of the following, depending on the sound source:

DESKMATE88_MUSIC - Parameters set for music compression

DESKMATE88_SPEECH - Parameters set for speech compression

DESKMATE88_ADJUSTABLE - Complete control of the parameters

The caller must keep track of the length of the uncompressed sound (which can be computed by $\text{end} - \text{start}$, or $\text{pSound} \rightarrow \text{sndlen} - \text{start}$ if end is 0) and carry it with the compressed sound for use on decompression. The length of the compressed sound (the sum of the return values from **snd_compress_part()**) must also be carried with the compressed sound for use when decompressing.

This algorithm has three parameters that adjust how the compression is performed. They are fields in the **COMPPARAM** structure: **precision**, **threshold**, and **threshold_length**. These fields are *only* used when the **compress_mode** field is set to **DESKMATE88_ADJUSTABLE**. If one of the other values for **compress_mode** is selected, predefined values for **precision**, **threshold**, and **threshold_length** are assumed.

When **compress_mode** is **DESKMATE88_MUSIC**, **precision** is 1 and **threshold** is 0 (when **threshold** is 0, it doesn't matter what **threshold_length** is set to, since setting **threshold** to zero disables method C [see the discussion below]).

When **compress_mode** is **DESKMATE88_SPEECH**, **precision** is 1, **threshold** is 12, and **threshold_length** is 60.

By selecting **DESKMATE88_ADJUSTABLE** these parameters can be used to adjust the level of compression, in exchange for certain trade-offs. These trade-offs consist of more or less sound quality for less or more compression, respectively. In order to understand how to adjust the

DeskMate Technical Reference Tandy Digital Sound Library

values, it will first be explained in general how the compression is done, how the variables affect the compression, and the gains and losses for each.

```
/* Compression-precision default defines */
SND_PRECIS      1      /* "precision" compression default */
SND_SPEECH_THR  12     /* "threshold" speech comp. default */
SND_MUSIC_THR   0      /* "threshold" music comp. default */
SND_THRESH_LEN  60     /* "threshold_length" default */
```

These defines are the default control parameters as available from sound.h.

The DESKMATE88 compression algorithm performs several different types of compression on the sound presented to it. These are:

- A) Scan the entire length of the sound and look to see which delta values occur more often than any others (delta values are explained below). Then, build a table of the fourteen most common values. Whenever the difference (or delta value) between the current sample and the next sample is in the table, use a four-bit quantity (sufficient to hold values from 0-13) to represent the new sample instead of the actual eight-bit sample.
- B) Replace long repetitions of the same sample value with a special code and a number telling how many times to repeat the value.
- C) Look for long repetitions of low level sound that might be pauses between words and replace this with silence (followed by an application of Method B).

Because Method B in no way affects sound quality, it has no parameters to change its behavior.

Methods A and C, however, are capable of actually removing some of the redundancy and unused or unwanted information in the sound to be compressed. Thus, parameters are available to alter their behavior. In the case of Method A, that parameter is precision. In the case of Method C those parameters are threshold and threshold_length.

DeskMate Technical Reference

Tandy Digital Sound Library

METHOD A

Let's say that we have the following group of sound samples from an imaginary sound (all numbers are given in hexadecimal):

80 84 86 78 74 70 89 74 80 85

On the screen of the DeskMate sound editor, sounds are displayed with 0 at the bottom of the screen, 80 at the middle (normal silence), and FF at the top of the screen. So, the data above would appear to oscillate above and below the center line displayed by the sound editor.

The difference (or delta value) between the first and second samples is 4 (84 - 80), between the third and fourth is -14 (78 - 86), and so on.

Delta values are valuable because most sounds use the entire spectrum of sample values (00 through FF) but the changes from sample to sample tend to be small and steady, thus the delta values all tend to be small and clustered around zero.

In a typical sound, fourteen delta values will account for 80% of the changes between one sample and the next. Therefore, a cache table can be built that contains the most common delta values and use only four bits to specify a cache table entry rather than eight bits to specify a new sample value.

precision is used to decide how close a delta value needs to be to an entry in the table of most common deltas, before we will use that entry. For example, to ensure that the decompressed sound is exactly like the original, precision is set to 0. This ensures that our current difference will *exactly* match one in the table before using that table entry. However, if a close match is sufficient (and it usually is for the human ear) then it might be decided that the current difference only has to match a delta table entry within one before using the table entry. In that case, precision is set to 1.

Precision is set to 1 by the DeskMate sound editor when it compresses both speech and music. Precision values must lie between 0 and 3 with 0 being exact reproduction of the original sound and 3 being rather significant distortion.

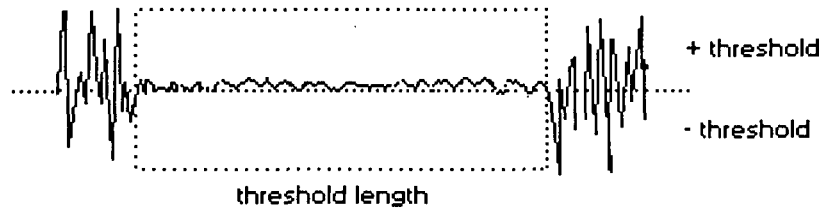
The best way to determine whether a precision of 1 or above will be acceptable on a particular sound is to try it on some similar sounds and listen to the results. A precision of 2 is fairly acceptable on speech and music with only a slight increase in "hiss" as the result.

METHOD C

Fortunately Method C is enormously easier to understand and adjust. Its two parameters specify a moving box that is centered around the "origin" or silence value of 80 (hex). This box moves forward, and if at any time the box encounters a region where *all* of the sound sample values fall within the box it assumes that this region contains only low level background noise (perhaps between spoken words) which may be replaced with silence.

DeskMate Technical Reference Tandy Digital Sound Library

The two parameters determine the size of the box (and what it will consider to be background noise) in the following way:



Thus, `threshold` determines the height of the box and `threshold_length` determines the length.

Setting the value of `threshold` to 0 will cause compression Method C to be turned off altogether. It is recommended that this method be turned off for anything other than speech, such as recorded music or sound effects. Setting it to higher values (like 12, which is the value used for speech compression within the DeskMate sound editor) will determine how loud the background noise can get and still be removed. Setting `threshold` at too high a value can cause low levels of speech to be removed completely, so care must be exercised in setting this value. Setting it too low would allow small rises in the background noise level to be considered as "signal", preventing compression (removal) of this part of the noise.

The value that `threshold_length` is set to is multiplied by 51 in order to determine the number of samples that must fall within the threshold boundaries before being replaced with silence. Setting this value too high would cause hiss to be ignored because a long enough pause of "noise" could not be found. Setting this value too low would cause the small pauses *within* words to be replaced with silence (resulting in a strange clipped sound to the speech). The value for this parameter is 60 when speech is compressed by the DeskMate sound editor.

DeskMate Technical Reference Tandy Digital Sound Library

Appendix B BIOS DIGITAL SOUND SUPPORT

The BIOS in the Tandy 1000 SL/TL and TL/2 computers has the same sound support sound as all previous Tandy 1000 computers and also has support for the new digital sound hardware. The interface for these new BIOS calls is defined as follows:

INT 1A

(AH) = 81H Get sound status

On return:

(AX)=00C4H

Not Busy:

(CF)=0

Busy:

(CF)=1

(AH) = 82H Input Sound (from the microphone)

On entry:

(ES:BX) = Buffer Address

(CX) = Buffer length

(DX) = Sampling Rate (8 - 4095, where 8 is the fastest supported rate)

On Return:

(AH)=00

(CF)=0 Not Busy

(CF)=1 Busy

(AH)=83H Output Sound (to the speaker)

On entry:

(ES:BX) = Buffer Address

(CX) = Buffer Length

(DX) = Sampling Rate (8 - 4095, where 8 is the fastest supported rate)

(AL) = Volume (0 - 7 where 0 = no sound)

On Return:

Same as AH = 82H

NOTE: The hardware is guaranteed to be accurate up to approximately 40 kHz (sampling rate = 8). Faster rates may produce inconsistent results from one machine to another.

(AH)=84H Stop Sound Input and Output

DeskMate Technical Reference

Tandy Digital Sound Library

Notes:

- The transfer rate values in register DX are not the same for calls AH=82H and AH=83H. To input a buffer of data with the AH=82H call with a given DX value and play it back with the AH=83H call so that it sounds the same, the DX value for output would have to be approximately 11.5 times as big as the DX value for input when run on a Tandy 1000 SL (8.00 MHz 8086 processor). This ratio is slightly different on other processors (8088 and 80286) and system architectures.
- These BIOS calls use the DMA hardware to input and output the sound buffer. When functions AH=82H and AH=83H are called, the BIOS initiates the I/O and returns to the calling program immediately. When the DMA transfer is complete, the BIOS receives a hardware interrupt and executes a software INT 15H with AH=91H and AL=0FBH. If an application program needs to know when the data transfer has been completed, it will have to hook INT 15H and watch for this event.
- These BIOS calls mask the hardware restriction of not being able to DMA across a 64K memory address boundary from the caller, but the sound must be less than or equal to 64K in size.

DeskMate Technical Reference Tandy Digital Sound Library

Appendix C DeskMate 03.02.0x & 03.03.0x Sound File Format

A sound file is divided into three parts: the Header, the Note Information Block, and the sound data itself.

The header is the same for all sound files and consists of the following:

<u>Length</u>	<u>Purpose</u>
---------------	----------------

Byte	X'1A' (MS-DOS end-of-file byte).
------	----------------------------------

Byte	Compression: 0 for uncompressed file. 1 for file compressed with "music" compression (threshold equal to 0). 2 for file compressed with "speech" compression (threshold equal to 12).
------	--

Byte	Count. The number of "notes" defined in this file. If this is not an instrument file, this will be 1.
------	---

Byte	Instrument number: 0 for plain sound files. 1-32 for instrument files when this field is set. X'FF' for instrument files that haven't had a number assigned yet.
------	---

10 bytes	Instrument name. Up to 9 bytes terminated by a 0 byte. Actually, a name can be assigned even to a plain sound file, but it has no practical purpose. For an instrument file, this shows up on DeskMate's Instrument listbox in Music 1988 and DeskMate 03.03.0x.
----------	--

Word	The sampling rate at which the file was originally recorded. For files recorded by the Sound Editor, this will be X'157C' (5500), X'2AF8' (11000), or X'55F0' (22000). An instrument file will always have X'2AF8' (11000).
------	---

The Note Information Block consists of one of the following entries for each note in the file. A plain sound file will only have one of these, while an instrument file may have up to 16.

<u>Length</u>	<u>Purpose</u>
---------------	----------------

Byte	Pitch. This is stored as a number from 1 (A1) to 63 (B6). If this value hasn't been set yet (for an instrument file), it will be X'FF'.
------	---

Byte	Reserved.
------	-----------

Byte	Low range. The lowest note that Music should use this note for. If not set, or not an instrument file, will be set to X'FF'.
------	--

Byte	High range. The highest note that that Music should use this note for. Set to X'FF', if not set or if plain sound file.
------	---

Long	Location in file. Offset from beginning of file to the actual sound data.
------	---

DeskMate Technical Reference

Tandy Digital Sound Library

Long Begin selection. Set to X'00000000'. Exception: in a compressed sound file, this holds the number of bytes actually used to store the sound in compressed format.

Long End selection. Set to X'00000000'.

Long End. The total number of samples in the sound. For a compressed sound, this is still the number of samples when the sound is uncompressed.

Long Begin sustain. The sample number from the start of the sound of the "sustain loop".

Long End sustain. The end of the sustain loop. If 0, the instrument has no sustain defined.

The sound data merely consists of a stream of bytes for each sound. The beginning and end of each sound is denoted only in the Note Information Block above. Note that X'80' represents a zero crossing (silence).

This format may only be read in future versions of the DeskMate Sound Editor.

DeskMate Technical Reference Tandy Digital Sound Library

Appendix D DeskMate 03.02.0x and 03.03.0x Environment Issues

There are several issues regarding the Tandy Digital Sound Library and the DeskMate 3 environment that the DeskMate developer must be aware of and carefully code. Otherwise the library calls may fail and cause the system to crash. An example of the correct handling of these issues is found in the DeskMate demonstration program on the distribution diskette.

Compiling

When compiling the DeskMate demo with Microsoft 'C' 5.0 the variable `__STDC__` should be defined on the compile line.

Task Switching and Accessories

The DeskMate application must observe the following rules:

1. Code shedding must be disallowed in order to protect pointers to sound buffers maintained by the sound library and other pieces of critical data.
2. When a Task Switch is recognized `snd_exit()` must be called prior to giving up control. When control is returned, `snd_init()` must then be called.
3. The memory allocated for the sound library must not be freed. The pointers to these memory areas are passed to the library via `snd_addbuf()`. The sound library assumes these buffers remain available across calls to `snd_exit()` and `snd_init()`.
4. Task Switch must not done while compress or decompress is in progress.

Joystick

As pointed out in the Introduction and Overview section, the same DAC is used by the joysticks and for both playing and recording. Therefore, joysticks are disabled while the sound library is in use. DeskMate applications must check to see if the joystick is the current pointing device. If so, then the joystick must be disabled. Refer to the file `DMDEMO.C` in the `\DESKMATE` directory on the distribution disk for an example of correctly checking for and disabling the joystick.

There are also certain DeskMate calls which will conflict with the Sound Hardware and therefore must be avoided when the sound library is enabled (`snd_init` has been called.) They are: any of the `ms_` calls, and `event_purge()`. To work around this problem the DeskMate demo contains the two calls `Set_Joy()` and `Un_Set_Joy()` which disable the interrupt 33 vector if the joystick driver is loaded, and resets the interrupt 33 vector when finished.

Tandy

Digital Sound

Tandy 1000TL

Tandy 1000SL 25-1401

Objective:

Show that Tandy has both the hardware and software tools that you need to enhance your software products with sound.

Outline:

- I. Digital Sound Hardware
- II. Concepts of Digital Sound
- III. Software design strategies
 - A. Direct to sound hardware
 - B. ROM BIOS
 - C. Sound Toolkit I
- IV. DeskMate Sound Editor
- V. Question and Answer

Digital Sound Hardware

PSSJ Custom IC

- PSSJ: Parallel/Serial/Sound/Joystick
- Backward compatible to:
TI Sound Chip used in all previous Tandy 1000s
and IBM PC Jr.

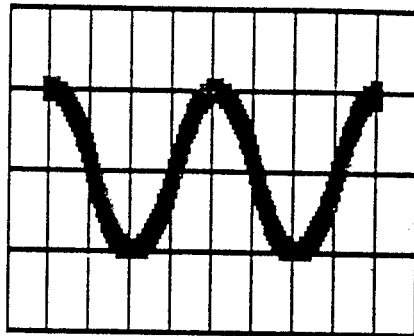
Timer-generated sound on 8253.
- Sampling rate range
80 samples per second minimum
100,000 samples per second maximum
- 3 bit attenuator
Adjust the volume at the speaker from no sound
at all (0) to maximum volume (7) which closely
approximates the TI sound chip.
- Mix TI sound and Digital Sound
Both TI sound and Digital sound may be
programmed to be on and both outputs will be
sent to speaker.

Digital Sound Hardware

- Digital-to-Analog Converter (DAC)
- Sound produced by sending different voltages at regular intervals to speaker.
- Successive Approximation
- Use DAC to approximate a sampled voltage.

Digital Sound Concepts

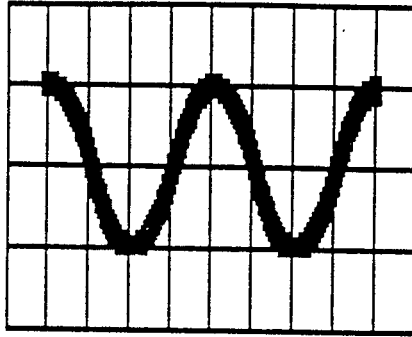
- A sample is a snapshot of an analog waveform.



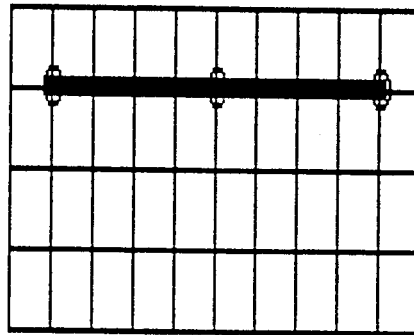
5.5 KHz Input Signal

- Samples occur at regular intervals along the waveform.

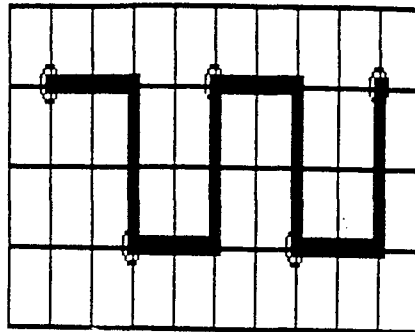
- The sampling rate is the frequency at which the analog waveform is sampled.



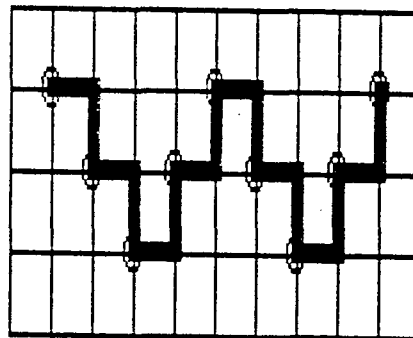
5.5 KHz Input Signal



Record / Playback at 5.5 KHz



Record / Playback at 11 KHz



Record / Playback at 22 KHz

- Quality of sound varies directly with sampling rate.

Software Design Strategies

A. Program directly to hardware

- Quite intricate

- Refer to Technical Reference Manual
Jacksboro Specification

B. ROM BIOS support for sound

- Simple interface
- Input a sound
 - Buffer address, Buffer length, Sampling rate
- Output a sound
 - Buffer address, Buffer length, Sampling rate
 - Volume control
- Invokes DMA and returns to program
- DMA complete signalled via INT 15h
- Sound size limited to 64K
- Documentation found in Technical Reference Manual (BIOS Sound Support)

-Use the BIOS to recognize a computer that has Tandy Digital Sound.

Use the following two tests:

1. Function AX=8100 and INT 0x1A
If AX < 0x8000 on return, then sound hardware may be present. The value returned in AX is the port number of the control register of the sound hardware on this particular machine.

If AX >= 0x8000 on return, then there is no BIOS support for digital sound, and thus no hardware.

2. If test 1 was successful, then perform the following test:

Two different test values must be written and then read back from Control Port + 2 (see #1). The tests must be in the sequence WRITE 1, READ 1, WRITE 2, READ 2. If both values read back correctly, sound hardware is available.

If either test fails, then the sound hardware is not present or it is malfunctioning.

Tandy Digital Sound Library

- Application Program Interface (API)
- High-level interface to record, play, and manipulate sounds.
- A toolkit, enabling the developer incorporate sound in an application.

Toolkit I Demo

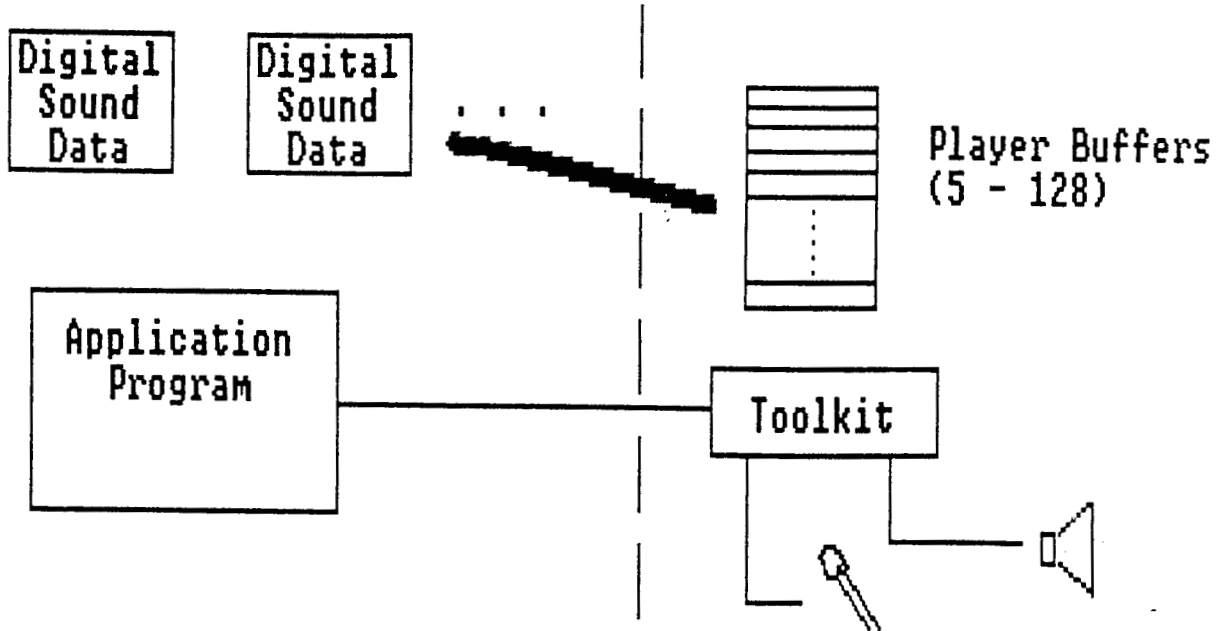
- Kept simple to exhibit toolkit functions.
- Records a sound and plays it back.
- Compresses a sound and writes the compressed sound to a file.
- Reads a compressed sound and decompresses it into memory.
- Uses 11 of 16 Toolkit I calls, including all major calls.

Why use Toolkit I ?

- Buffer Queueing
- Click Minimization
- Sound buffers larger than 64K
- Quad Sampling
- Animation Support
- Compress/Decompress algorithms available
- **Source for demonstration programs**
Demonstration programs for both DeskMate and non-DeskMate applications.

Toolkit I Overview

- Toolkit I linked to application
- Adds 9K to application if both sound drivers and compression/decompression routines are used. Adds 4K if sound drivers only are used.
- At least 5 2K buffers are required for the player mechanism.
- Room in memory is required for your sound data in uncompressed format.
- Microsoft C calling conventions
- Toolkit I allocates no memory.
This enables application to choose the memory allocation scheme appropriate to the application.

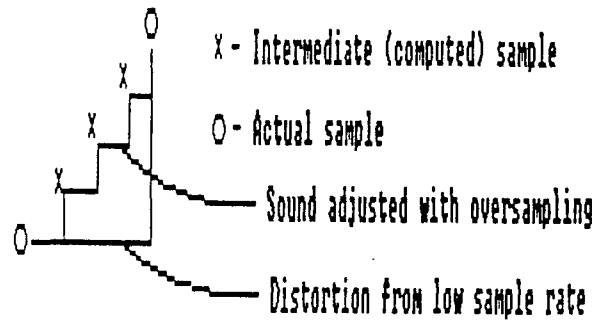


Toolkit Features

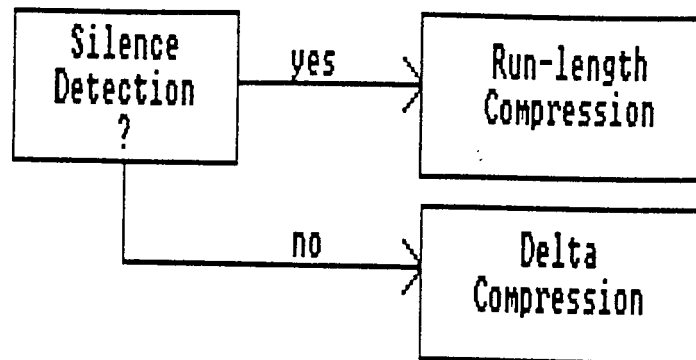
- Sounds greater than 64K
Sounds broken up in small chunks, but played and recorded contiguously.
Sounds must fit uncompressed in memory.
- Individual sounds queued to player
Sounds can be very small.
- Animation
Player keeps track of sounds in the queue.
Application paused with call to toolkit
Control returned to application when sound starts
- Automatic detection of presence/
non-presence of sound hardware
Follows the recommended BIOS method for
determining the presence of sound hardware

Toolkit Features

- Distortion reduction by oversampling
Used when sampling rate is 11kHz or 5.5kHz
Player always operates at 22kHz
Intermediate samples are computed so that 22,000 samples per second are still produced.



- Built-in Compression/Decompression



Typical Compression Ratios:

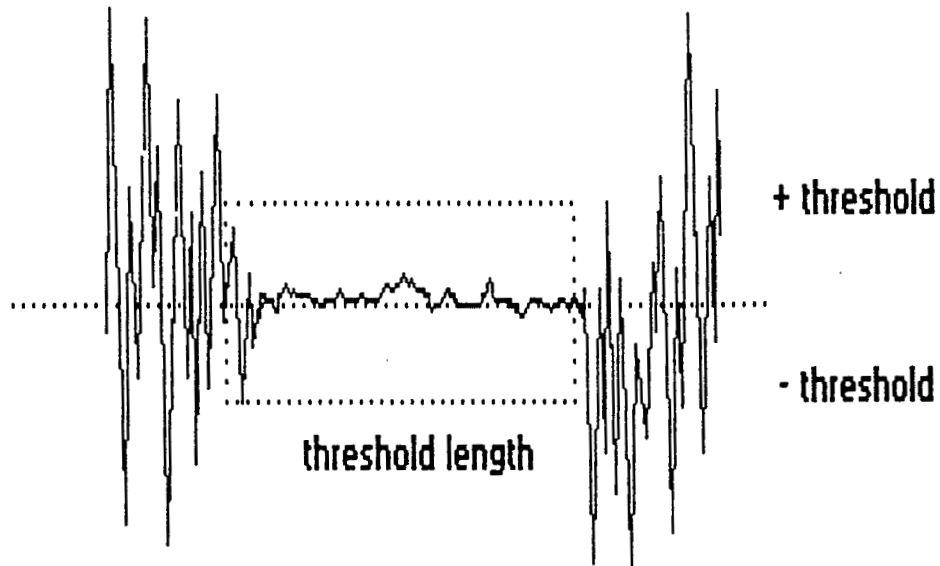
Speech - approximately 40%

Music - best case is 50%

Compression Techniques

Silence Detection

- Background noise is low-amplitude samples
- Such samples are set to hexadecimal 80, and then compressed out by run-length encoding.



- Size of box adjustable by changing threshold and threshold_length.

threshold - controls how near samples must be to silence in order for them to be compressed out as silence.

threshold_length - determines how many consecutive samples in the threshold constitute background noise.

Caveats

- If threshold too high, low levels of speech may be removed.
- If threshold_length too high, background noise may remain.
- If threshold_length too low, small pauses within words may be removed.

Compression Techniques

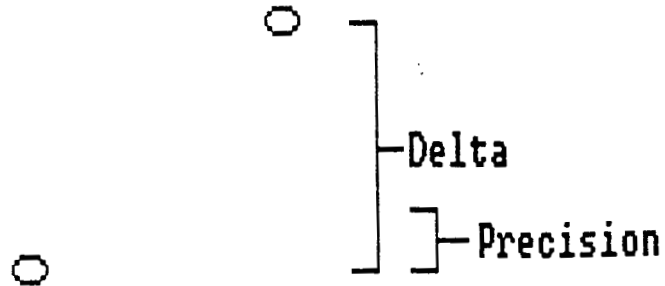
Run-length Encoding

- Repeated samples compressed out.

Delta Encoding

- In digitized sounds, changes from sample to sample tend to be slow and steady.
- Pass 1 of compress establishes a table of the most common deltas.
- Samples whose delta occurs in the common delta table are stored as an offset into the table in four bits, rather than the eight-bit sample.
- Common delta table catches typically 80% of samples.

- The precision parameter specifies how near a delta must be to be stored as a common delta table offset.



precision=0 exact match on compress; exact reproduction on decompress

precision=3 aggressive on compression; significant distortion on decompress

Recommended compression tuning parameters:

Music - precision = 1
 threshold = 0

Speech - precision = 1
 threshold = 12
 threshold_length = 60

Toolkit I Functions

snd addbuf

Assigns memory to the player's buffer pool.

Parameters:

- far pointer to a buffer
- number of buffers pointed to

snd init

Initializes the sound chip, disables the joysticks, sets options, and prepares to play or record.

Parameters:

- number of buffers accumulated before player starts
- DMA channel to use. Defaults to channel 1
- Interrupt channel to use. Default is channel 7
- FASTRAMP option. Overrides click suppression

snd exit

Halts playing or recording if necessary, disables the sound chip and reenables the joystick. Interrupt vector used is restored to original value.

No parameters.

snd cue

Places the sound driver into standby mode for recording or playing.

Parameters:

- Sound driver mode. Specifying TORECORD puts the driver in standby mode for recording. Specifying TOPLAY puts the driver in standby mode for playing.

snd record

Records a sound into one or more buffers.

Parameters:

- Far pointer to sound data structure
- Maximum record time
- Sampling rate to use
- Wait for standby buffer option

snd play

Queues a sound for playing.

Parameter:

- Far pointer to sound play structure

snd flush

Informs the player that there are no more sounds queued to be played in this group, and it is to immediately start playing what has been queued so far.

Parameters:
(none)

snd stop

Stops the sound output or stops recording.

Parameters:
- Far pointer to sound play structure

Can stop at a specified sound, or stop immediately, regardless of location.

snd wait

Provides status on the player, recorder, or a specific sound in the play queue (possibly waiting for the specified sound, the player, or the recorder to finish) and then returns.

Parameters:

- Far pointer to a sound play structure
- Wait flag

Various parameter combinations permit an application to:

- Wait to within 100msec of a particular sound to begin.
- Wait for all sounds to complete playing or all recording buffers to be filled.
- Check if a particular sound is about to be played.
- Check if the player or recorder is running.

snd_amplify

Generates a function table to be used to amplify (or de-amplify) a sound by the specified factor.

Parameter:

- Amplification factor

snd_compensate

Generates a translation table which can be used to adjust the recorded digital data to be centered around the digital centerline (hexadecimal 80)

Parameter:

- Far pointer to a 256 byte table.

snd_apply_func

Applies the function represented by a table to a portion of a sound.

Parameters:

- Far pointer to a sound data structure
- Offset in sound where manipulation is to begin
- Offset in sound where manipulation is to end
- Far pointer to 256 byte table

snd_compress part

Performs data compression.

Parameters:

- Far pointer to a sound data structure
- Far pointer to a buffer to receive compressed data
- Size of compressed data buffer
- Type of compression algorithm to be used
- Far pointer to compression data structure

snd_decompress part

Decompresses a previously compressed sound.

Parameters:

- Far pointer to a sound data structure
- Far pointer to a compression data structure.
- Far pointer to a buffer of compressed data
- Length of compressed data

snd volume

Sets the hardware gain level for the next time the player is started.

Parameter:

- Volume setting. Ranges from 0 (no sound output) to 7 (maximum volume setting)

snd version

Returns version information from the toolkit.

Parameter:

- Far pointer to a version information structure

Proposed Toolkit II Enhancements

- Record to hard disk file
- Play from hard disk file

Sounds can be much longer than available memory
Reduces the need for sound buffers in memory

- Optional real-time sound compression on record, Optional real-time sound decompression on playback.
- Background-like behavior for file play and file record functions.

Play or record starts running and control is returned to the application.

- DeskMate Resource

DeskMate Sound Editor

- Records and Plays Sounds
- Edit out extraneous sounds or unwanted pauses
- Manipulate sounds via Cut/Paste
- Amplify sections
- Compress and Decompress sounds
- Choose from sampling rates of 5.5kHz, 11kHz or 22kHz.
- Useful tool to prepare sound files for use with your application.
- Comes included with DeskMate on the Tandy 1000SL and 1000TL.

Demo Program

Purpose:

Write a program that records a sound and plays it back.

Steps Required:

1. Obtain player buffers.
2. Setup the sound hardware.
3. Obtain buffers for sound data.
4. Record a sound.
5. Play the sound back.


```

/*
 * "sound.h" - Include file for use with Tandy digital sound library
 *              as of Version 1(10)
 */

typedef unsigned long    u_long;
typedef unsigned int     u_int;
typedef unsigned short   u_short;
typedef unsigned char    u_char;

typedef char far *       saddr;

typedef struct sound {
    saddr          buffer;          /* The sound buffer itself */
    u_long         sndlen;          /* Length of the sound in the buffer */
    u_long         buflen;         /* Length of the buffer itself */
    u_char         rate;           /* Rate the sound was recorded at */
    u_char         system;        /* Sound source (see defines below) */
    u_long         reserved[4];    /* Reserved for internal use */
} SOUND;

typedef struct sndhdr {
    /* The following items, except "ticket", "reserved1",
       and "reserved2" are to be filled in by the caller */
    SOUND far *sndp;              /* The sound structure */
    u_long         start;         /* Where to start playing (0 == ALL) */
    u_long         end;          /* Where to stop playing (0 == end) */
    u_short        ticket;       /* When buffer was/is-to-be played */
    u_char         rate;         /* Rate the sound is to be played at */
    u_char         reserved1;    /* Reserved for internal use */
    u_long         reserved2[4]; /* Also reserved for internal use */
} SNDHDR;

/* Machine sound came from */
#define B_SYNTH 0                /* Sound was synthetically produced */
#define B_MAC  0                /* Sound came from a Macintosh(R) */
#define B_AT8   1                /* Recorded on a 8mhz AT compatible */
#define B_SL    2                /* Sound was recorded on a 1000-SL */
#define B_TL    3                /* Sound was recorded on a 1000-TL */

/* Sampling rate defines */
#define R5500   1                /* 5.5khz sample rate */
#define R11000  2                /* 11khz sample rate */
#define R22000  3                /* 22khz sample rate */

typedef struct versioninfo {
    char far *    version;        /*Version format: Major[Minor](Edit)*/
                                /*String containing version message
                                   that can be displayed via printf()*/
    u_char        major;         /*Major version number*/
    u_char        minor;        /*Minor version number 0=no character,
                                   1=A, 2=B, 3=C...*/
    u_short       edit;         /*Edit number*/
    u_short       reserved1;    /*Reserved for future use*/
    u_long        reserved2;    /*Reserved for future use*/
}

```

```

        u_long          reserved3;          /*Reserved for future use*/
    } VERSIONINFO;

/* Scratch area for internal use of compress/decompress routines */
typedef struct {
    u_long  scratch[25];          /* Scratch area */
} COMPINFO;

/* Defines for compress "type" parameter */
#define CTYPE_DESKMATE88        0

/* pParam structure for compressing with type=CTYPE_DESKMATE88 */
typedef struct compparam
{
    COMPINFO far *pinfo;          /* compress/decompress work area */
    u_long  start;                /* start position in sound */
    u_long  end;                  /* end position in sound (0 = to the end) */
    u_short compress_mode;        /* what kind of compression to do */

    /* following parameters used only by */
    /* compress_mode = DESKMATE88_ADJUSTABLE */

    u_short threshold_length;     /* minimum length of "silence" */
    u_char  threshold;            /* amplitude of "silence" */
    u_char  precision;            /* how accurate to make deltas */
} COMPPARAM;

/* Defines for compparam "mode" parameter */
#define DESKMATE88_MUSIC        1          /* music compression */
#define DESKMATE88_SPEECH       2          /* speech compression */
#define DESKMATE88_ADJUSTABLE   3          /* roll-your-own compress mode */

/* defines for snd_cue() "mode" parameter */
#define TOPLAY                  1          /* prepare to play */
#define TORECORD                2          /* prepare to record */

/* defines for snd_init() "options" parameter */
#define FASTRAMP                0x0001     /* fast change between record and play */
/* Misc defines */
#define TRUE                    1
#define FALSE                   0
#ifdef NULL
#define NULL                    0
#endif /* NULL */

/* Error codes */
#define BADFMT -3                /* Decompress called with corrupted data
                                or sound compressed with unsupported
                                algorithm */
#define BADPARM -2              /* Compress called with bad type or
                                compress_mode */
#define INVALID -1              /* Call made when driver uninitialized
                                or call not legal at this time */
#define NOERROR 0

```

```

#define INITBUF 1 /* Insufficient buffers were allocated
                  before snd_init() was called */
#define INIT64K 2 /* All buffers spanned 64K physical
                  addresses - player cannot initialize */
#define BADDMA 3 /* DMA Channel requested is invalid */
#define BADIRQ 4 /* The IRQ number requested is invalid */
#define NOIRON 5 /* Can't find our hardware */
#define BADSIZE 6 /* Record buffer is too small or
                  too small for the requested time. */
#define WOULDBLOCK 7 /* snd_record() failed because two buffers are
                     already queued and block was set to FALSE */

/* Compression-precision default defines */
#define SND_PRECIS 1 /* "precision" compression default */
#define SND_SPEECH_THR 12 /* "threshold" speech comp. default */
#define SND_MUSIC_THR 0 /* "threshold" music comp. default */
#define SND_THRESH_LEN 60 /* "threshold_length" default */

/* Prototypes for routines */
/* __STDC__ is automatically defined for compilers supporting ANSI function */
/* prototypes, and undefined for compilers that don't. An ANSI and non-ANSI */
/* prototype is given for each function. */

#ifdef __STDC__
short far snd_addbuf(saddr mem, u_short nr_bufs);
short far snd_init(u_short trigger, u_short dma_ch, u_short irq,
                  u_short options);
short far snd_exit(void);
short far snd_play(SNDHDR far *pSound);
void far snd_volume(u_short volume);
short far snd_flush(void);
short far snd_record(SOUND far *pSound, u_short maxTime, u_short rate,
                    u_char far *adj_table, u_short block);
short far snd_wait(SNDHDR far *pSound, u_short wait);
void far snd_stop(SNDHDR far *pSound);
void far snd_amplify(u_short factor, u_char far *output);
void far snd_apply_func(SOUND far *pSound,
                      u_long start, u_long end,
                      u_char far *table);
long far snd_compress_part(SOUND far *psound, u_char far *dest,
                          u_short dest_size, u_short type, COMPPARAM far *pParam);
short far snd_decompress_part(SOUND far *psound, COMPINFO far *pinfo,
                             u_char far *src, u_short src_len);
short far snd_compensate(u_char far *adj_table);
short far snd_cue(u_short mode);
VERSIONINFO far * snd_version(void);
#else /* __STDC__ */
short far snd_addbuf();
short far snd_init();
short far snd_exit();
short far snd_play();
void far snd_volume();
short far snd_flush();
short far snd_record();

```

```
short far snd_wait();
void far snd_stop();
void far snd_amplify();
void far snd_apply_func();
long far snd_compress_part();
short far snd_decompress_part();
short far snd_compensate();
short far snd_cue();
VERSIONINFO far * far snd_version();
#endif /* __STDC__ */
```

```
# include "sound.h"
# include "dsetup.h"
# include "dgetbuf.h"
# include "drecord.h"
# include "dplay.h"
# include "demo1.h"
/*
 *      This demonstration program records a sound and plays it back.
 */

# ifdef __STDC__
int      main(void)
# else
int      main()
# endif
{
    /* get buffers for the sound library, and set things up */
    set_up_sound((u_short) 5, (u_short) 0, (u_short) 0, (u_short) 0,
                 (u_short) 0);

    /* allocate memory to store the sound in (as much as possible) */
    set_up_buffers();

    /* record the sound */
    do_record_sound();

    /* play it back */
    do_playback_sound();

    /* shut down the hardware */
    snd_exit();

    /* exit */
    return 0;
}
```

```
# include <stdio.h>
# include <malloc.h>
# include <stdlib.h>
# include "sound.h"
# include "dsetup.h"

/*
 * Set up for using the sound library, getting the buffers using
 * far malloc.
 *
 * nobufs - number of buffers to allocate for sound library use
 *          The minimum value of nobufs is 4 (5 for continuous
 *          play).
 * trigger - number of buffers to fill before starting playback
 *          (0 for maximum possible buffers)
 * dma_channel - which dma channel to use.
 *              (u_short) 0 indicates a default.
 * irq - which interrupt channel to use.
 *         (u_short) 0 indicates a default.
 * options - various option flags to be passed to snd_init.
 *
 * See the snd_init() documentation for more info on the last 4
 * parameters.
 */
# ifdef __STDC__
void far set_up_sound(u_short nobufs, u_short trigger, u_short dma_channel,
u_short irq, u_short options)
# else
void far set_up_sound(nobufs, trigger, dma_channel, irq, options)
u_short nobufs, trigger, dma_channel, irq, options;
# endif
{
    int      bufno;          /* buffers allocated counter */
    int      len;            /* appropriate size for buffers */
    char far *cp;            /* pointer to last allocated buffer */
    int      ret;            /* snd_init return code */
    char      versbuf[100];   /* buffer for version string */
    char far *vp;            /* pointer into version string */

    /* Print the version number */

    for (vp = snd_version()->version, cp = versbuf; *vp; vp++, cp++)
        *cp = *vp;
    *cp = 0;
    printf("Sound Library %s\n", versbuf);

    /* find out what size buffer the sound library wants */
    len = snd_addbuf((SOUND far *) 0, (short) 0);

    /* loop for however many buffers are needed */
    for (bufno = 0; bufno < nobufs; bufno++)
    {
        /* allocate a buffer in far data space */
        cp = _fmalloc(len);
        /* check if allocation was successful */
    }
}
```

```

        if (cp == NULL)
        {
            fprintf(stderr, "Cannot allocate sound buffers\n");
            exit (1);
        }
        /* give the sound library 1 buffer */
        snd_addbuf(cp, (short) 1);

        /* loop for the next buffer */
    }
    /* all the buffers are allocated */

    /* initialize the sound library */
    ret = snd_init(trigger, dma_channel, irq, options);
    switch(ret)
    {
        case NOERROR:    /* ok */
            return;
        case INITBUF:    /* insufficient buffers */
            printf("snd_init:  insufficient buffers\n");
            break;
        case INIT64K:    /* buffers span 64k boundary */
            printf("snd_init:  all buffers span 64k boundary\n");
            break;
        case BADDMA:     /* DMA channel requested is invalid */
            printf("snd_init: DMA channel requested is invalid\n");
            break;
        case BADIRQ:     /* IRQ number requested is invalid */
            printf("snd_init:  IRQ number requested is invalid\n");
            break;
        case NOIRON:     /* Can't find sound hardware */
            printf("snd_init:  Cannot find sound hardware \n");
            break;
        default:         /* unknown error code */
            printf("snd_init:  unknown error code %d\n", ret);
            break;
    }
    exit(1);
}

```

```
# include <stdio.h>
# include <malloc.h>
# include "sound.h"
# include "dgetbuf.h"

/*
 * This routine allocates as many sets of SNDHDR and SOUND structures,
 * and memory for the sound data. It grabs all the memory it can, so
 * be sure that sound library buffers are already allocated before
 * calling this routine.
 */

SNDHDR far * sndheaders[MAXBUFFERS]; /* list of SNDHDR structures */
SOUND far * sndstructs[MAXBUFFERS]; /* list of SOUND structures */

/*
 * actually, we can't possibly get 10 64k buffers. (64k * 10 = 640k,
 * and that doesn't include the SNDHDR and SOUND structs, or the program,
 * or MS-DOS). On a 640k machine, you can often get 7.
 */
int numbuffers = 0; /* number of buffers actually allocated */

# ifdef __STDC__
void set_up_buffers(void)
# else
void set_up_buffers()
# endif
{
    SNDHDR far *shp;
    SOUND far *stp;
    char far *bp;

    /*
     * Allocate as many BUFFERSIZE-byte buffers, and associated
     * SNDHDR and SOUND structs, as possible.
     */
    for (numbuffers = 0; numbuffers < MAXBUFFERS; numbuffers++)
    {
        bp = _fmalloc(BUFFERSIZE);
        shp = (SNDHDR far *) _fmalloc(sizeof(SNDHDR));
        stp = (SOUND far *) _fmalloc(sizeof(SOUND));
        /* check if any allocations failed */
        if (bp == 0 || shp == 0 || stp == 0)
        {
            /* an allocation failed - free any pieces that
             * were allocated and can't be used */
            if (bp != 0)
                _ffree(bp);
            if (shp != 0)
                _ffree(shp);
            if (stp != 0)
                _ffree(stp);
            printf("Allocated %d %dK buffers\n",
                numbuffers, (BUFFERSIZE+512)/1024);
            return;
        }
    }
}
```



```
/* fill in the SNDHDR structure */
shp->sndp = stp; /* fill in pointer to SOUND */
shp->start = 0; /* fill in start/end */
shp->end = 0; /* 0 for both indicates the entire */
/* sound should be played */
shp->rate = R11000; /* Sampling rate is one of */
/* R5500, R11000, or R22000 */

/* fill in SOUND structure */
stp->buffer = bp; /* fill in buffer address */
stp->sndlen = 0; /* initially no sound recorded */

/* Lie about the buffer size, so there is enough slop */
/* at the end of the buffer so snd_decompress_part can */
/* decompress a full buffer back into the same-sized */
/* buffer without running off of the end. */

stp->buflen = BUFFERSIZE - 2; /* fill in buffer length */

/* save addresses so we remember where they are */
sndheaders[numbuffers] = shp;
sndstructs[numbuffers] = stp;
```

}

}

```
# include <stdio.h>
# include <conio.h>
# include "sound.h"
# include "drecord.h"
# include "dgetbuf.h"

/*
 *      This routine records a sound into the memory allocated by
 *      set_up_buffers().
 */

# ifdef __STDC__
void do_record_sound(void)
# else
void do_record_sound()
# endif
{
    int      i;
    int      prompted = 0;

    /* set up for recording */
    snd_cue(TORECORD);
    /* prompt the operator and wait for response */
    printf("Press ENTER to start recording\n");

    while (getch() != '\r')          /* wait for carriage return */
        ; /* do nothing */

    /* for each buffer */
    for (i = 0; i < numbuffers; i++)
    {
        /*
         * maxtime = 0 means "record until buffer fills up"
         * rate = R11000; use 11,000 samples/second
         */
        do
        {
            /* first pass through loop, give instructions */
            if (prompted == 0)
            {
                printf("Press any key to stop recording\n");
                prompted = 1;
            }

            /* Check for keyboard input, which is considered a */
            /* request to stop recording */
            if (kbhit())
            {
                /* stop recording */
                snd_stop((SNDHDR far *) 0);
                /* read character and throw it away */
                getch();
                printf("Recording stopped by operator request\n");
                return;
            }
        } while (snd_record(sndstructs[i], 0, R11000,
```

```
        (u_char far *) NULL, FALSE) == WOULDBLOCK);  
    }  
    snd_wait((SNDHDR far *) 0, (short) 1);  
        /* wait for recording to finish */  
    printf("Recording stopped - buffers full\n");  
    return;  
}
```

```
# include <stdio.h>
# include <conio.h>
# include "sound.h"
# include "dgetbuf.h"
# include "dplay.h"

/*
 *      This routine plays back a sound in the memory allocated by
 *      set_up_buffers() and loaded by something.
 */

# ifdef __STDC__
void do_playback_sound(void)
# else
void do_playback_sound()
# endif
{
    int      i;
    SNDHDR far * shp;

    /* prompt operator and wait for response */
    printf("Press ENTER to start playback.\n");
    while (getch() != '\r')          /* wait for carriage return */
        ; /* do nothing */

    /* for each buffer */
    for (i = 0; i < numbuffers; i++)
    {
        shp = sndheaders[i];      /* get SNDHDR struct address */
        if (shp->sndp->sndlen > 0) /* if there is sound in
                                   the buffer */
        {
            /* play sound at same rate it was recorded */
            shp->rate = shp->sndp->rate;

            /* start and end in the SNDHDR struct have already */
            /* been set to zero (play entire buffer) */
            /* play this buffer */
            snd_play(shp);
        }
    }
    snd_wait((SNDHDR far *) 0, (short) 1); /* wait for playing to finish */
    return;
}
```

Tandy® Digital Sound Library

Tandy 1000SL 25-1401
Tandy 1000TL

TEM-0002

Tandy Digital Sound Library Manual
Copyright 1989, Tandy Corporation
All Rights Reserved.

While reasonable efforts have been made in the preparation of this manual to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors in or omissions from this manual, or from the use of the information contained herein.

Tandy and DeskMate are registered trademarks of Tandy Corporation.

Microsoft is a registered trademark of Microsoft Corporation.



Tandy Electronics

A DIVISION OF TANDY CORPORATION

RESEARCH AND DEVELOPMENT
1300 Two Tandy Center
Fort Worth, TX 76102

To: **The Developer**

From: **Scott Cutler, Vice-President, Software Design, Tandy Computers**

Welcome! By using the Tandy Digital Sound Library, you join a growing number of software developers who are discovering the value of adding sound to their software products. This digital sound technology is available exclusively on Tandy 1000SL and Tandy 1000TL computers.

This library contains high-level functions you can call to record, manipulate and play sounds easily. It is a programmer's toolkit designed to be programmer-friendly. There are many details involved in using the digital to analog converter, the DMA, and so forth. All of this is handled for you in the sound library in a clean, compact way. The entire library is less than 10Kbytes, easily worth the space for the functionality you receive.

We suggest you start by carefully reading the "Introduction and Overview" beginning on page one. It contains an excellent introduction to the concept of digital sound, how it is implemented on Tandy computers, and the library calls available to you. You'll want to carefully read the next section where the data structures used by each of the calls are defined. Thorough familiarity with this section will save you much time debugging your digital sound application. The next step is to become familiar with the demonstration programs found on the distribution diskette. For DeskMate developers, a demonstration DeskMate application is included in the \DESKMATE directory. These programs enable you to see how the sound library calls are used, and how the various calls fit together in a complete digital sound application. Following the structures and defines section is complete documentation on each library call. The appendixes contain information you may need in certain situations.

Once again, welcome, and good luck programming with the Tandy Digital Sound Library!

Tandy Electronics telephone support:

For DeskMate software developers: 817/878-4845 or 817/390-2623

For all others: 817/878-6431

Tandy Digital Sound Libra., Technical Reference

Addendum

The **adj_table** parameter to **snd_record()** is ignored in this release of the Tandy Digital Sound Library. All subsequent releases of the Sound Library will honor this parameter.

In the meantime, you can achieve the same results by calling **snd_compensate()** to generate the table, perform the recordings with one or more calls to **snd_record()**, and then call **snd_apply_func()** to adjust the recorded sounds. A call to **snd_apply_func()** must be made for each buffer filled by **snd_record()**.

If the workaround described above is used, **snd_record()** should be called with the **adj_table** parameter equal to **(char far *)NULL**. By doing this, the code will continue to function as intended with subsequent releases of the Sound Library, when the **adj_table** parameter will be honored. Otherwise, the adjustment will be performed twice.

Table of Contents

Introduction	1
Hardware	1
Software	3
Distribution Disk Description	5
Structures/Defines	7
snd_addbuf	Adds memory for the player to use.....11
snd_amplify	Generates function table to amplify sound.12
snd_apply_func	Applies a function table to a sound.....13
snd_compensate	Creates a recording translation table.....14
snd_compress_part ...	Performs data compression on a sound.....15
snd_cue	Puts hardware in standby for play/record.16
snd_decompress_part .	Performs data decompression on a sound....17
snd_exit	Turns the sound chip off.....18
snd_flush	Starts playing queued sounds.....19
snd_init	Initializes the sound chip.....20
snd_play	Plays sound(s) in memory.....22
snd_record	Records a sound into memory.....24
snd_stop	Stops the sound output or stops recording.26
snd_version	Returns the version number.....27
snd_volume	Sets the output level for play.....28
snd_wait	Waits for sound(s) to finish.....29
Appendix A	Compression/Decompression.....30
DeskMate 88	DeskMate Sound 1988 compression.....31
Appendix B	Bios Digital Sound Support.....35
Appendix C	DeskMate Sound 1988 File Format.....37
Appendix D	DeskMate Environment Issues.....39

Tandy Digital Sound Library Technical Reference

Introduction and Overview

The Tandy® Digital Sound Library is for use with machines with built-in digital sound hardware. These machines are:

Tandy 1000 SL	cat. no. 25-1401
Tandy 1000 TL	cat. no. 25-1601

The digital sound hardware that is incorporated into the Tandy 1000 SL and TL can be used to record audible sounds and store them in digital form. These recorded sounds can then be stored on disk, transmitted by modem, or manipulated and combined with other recorded sounds. These sounds can then be "played back" by converting the digital information back into audible sounds. The Sound Library is designed to hide the complexities and requirements of the digital sound hardware from the software developer. It provides a set of functions that can be used to record, play, and manipulate sounds.

The Hardware

The digital sound hardware is an integral part of the Tandy 1000 SL/TL joystick circuitry. The key component is an eight-bit digital-to-analog converter, also known as a DAC. This device converts an eight-bit number into a corresponding voltage. Each bit that is turned on produces a specific amount of voltage. For example, Bit 1 produces twice as much voltage as Bit 0, Bit 2 produces twice as much voltage as Bit 1, and so on. The sum of all 8 bits (on or off) becomes the voltage that is amplified and sent on to the speaker. By varying the voltage output at regular intervals, an audible sound can be produced.

The method used to quantize, or digitally record, an analog sound is called Successive Approximation. When an input signal is to be digitized, several events occur. First, the voltage on the input is latched internally. This is done to prevent any signal variation from affecting the approximation of this sample. Then the same DAC that is used to create a sound begins generating eight different voltage levels by turning on and off bits in the DAC circuitry. The hardware determines if the voltage being input is equal or greater than the voltage the DAC outputs when a given bit is on. The most significant bit (7) is checked first to see if its level is greater than or equal to the level that was latched. If so, Bit 7 remains on and will be on in the finished byte. Otherwise, Bit 7 is turned off. Now Bit 7 is left as it is and the same type of comparison is done on Bit 6 and then on the remaining bits. The procedure resembles a binary search, in that each bit that is examined eliminates half of the remaining possible input signal levels. When all eight bits have been computed, the CPU can read the eight bit value present in the DAC and store it as one sample of the digital recording.

Since the same DAC is used by the joysticks and for both playing and recording, only one function can be done at a time. Therefore, the joysticks are disabled while the sound library is in use. (See `snd_init()` and `snd_exit()` in the reference section.)

To actually record or play back an audible sound, the record or playback hardware must obtain multiple samples by repeating these operations thousands of times a second. At the same time, software must manage the digitized data and get it to and from the hardware. Because it takes a minimum of two samples to reproduce a single cycle of an analog waveform, the highest frequency that can be recorded and reproduced is about one-half the sampling rate. This table can be used as a guide:

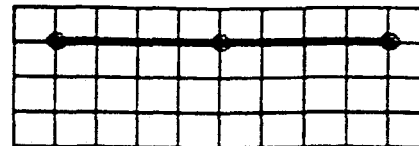
Tandy Digital Sound Library Technical Reference

<u>Sampling Rate</u>	<u>Highest Recordable Frequency</u>
5,500 Hz	2,500 Hz
11,000 Hz	5,000 Hz
22,000 Hz	10,000 Hz

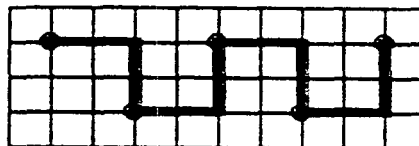
The higher the sampling rate, the higher the frequencies that can be recorded, while the lower frequencies will be recorded more accurately. For example, the following charts show a 5,500 Hz sine wave that is recorded at these three sampling rates and what the signal will look like in digitized form. Each sampling point is circled.



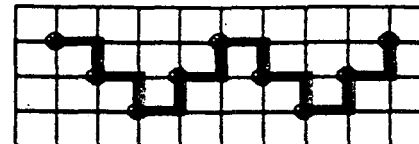
5.5 kHz Input Signal



Record / Playback at 5.5 kHz



Record / Playback at 11 kHz



Record / Playback at 22 kHz

If the sampling rate is too low, there is not enough information to produce something that sounds like the original sound. As the sampling rate increases the resulting waveform resembles the original more closely. A roll-off filter in the input hardware eliminates frequencies that could not be recorded even with the 22,000 Hz sampling rate. (A roll-off filter allows frequencies below a certain point to pass unchanged. Above that point, the higher the frequency, the less of the signal is allowed to pass.) Without this filter, significant distortion could be introduced into the recordings.

One specific form of distortion that arises in digital sound reproduction is called "aliasing". Most people have seen a form of aliasing when watching a movie where wheels that are in motion appear to turn backwards or stand still. The wheel appears to stand still because the wheel was turning at the same speed that the camera was exposing the film, and so the camera exposed each frame of the film when the wheel was in the same position. Aliasing in digital sound occurs when a frequency to be recorded is equal or greater than half the sampling rate. When this occurs, the resulting recording will not reflect the frequency of the input signal. For example, the chart above showing the result of the 5.5 kHz signal being sampled at 5.5 kHz shows no change in the recorded signal. So because each sampling occurred when the signal was at the same spot, the signal appeared to stand still. The input roll-off filter helps reduce this and most other forms of aliasing in spurious high-frequency signals, but the sampling rate must still be set high enough to record the highest fundamental frequency expected.

The output section also has a roll-off filter to reduce high frequency distortion created in the D to A process as well as switching noise caused by the processor.

To improve the resulting reproduction of a digitized sound recorded at a sampling rate lower than 22 kHz, the software computes intermediate samples so that 22,000 samples per second are still produced. This is called *oversampling*. At 11,000 Hz, one sample must be computed for every "real" one. At 5,500 Hz, three intermediate samples are computed. The intermediate samples

Tandy Digital Sound Library Technical Reference

are computed by taking the average between "real" samples. These extra samples result in a form of curve-smoothing being performed by making the change in amplitude between samples smaller. This reduces distortion in the output circuitry which appears when frequencies approach the limit of that sampling rate.

The Software

The functions (calls) provided in the Tandy Sound Library allow the caller to record and play sounds without having to deal with most of the hardware-related activities. Some of the key functions included in the sound library are discussed briefly below, and are discussed in more detail in later sections.

All functions included in the sound library are written to expect Microsoft® C-style calling rules and all functions should be called FAR. The library functions further assume that SS is equal to DS. Only the registers SI and DI are preserved across library calls. All functions return result codes in AX (short), unless otherwise specified. Functions returning a long do so with the most significant word in DX and the least significant word in AX. A few functions return far pointers with the segment in DX and the offset in AX. All of these return formats are compatible with Microsoft C.

Since the developer may choose to use a different language than C, the developer's program is responsible for allocating all memory required by the sound library functions. The program informs the sound library of the address of most of these memory areas by calling `snd_addbuf()`. If the language requires the memory to be released before terminating, the developer's code is also responsible for this task after calling `snd_exit()`.

The sound drivers within the sound library make extensive use of interrupts and one DMA channel to operate the sound hardware. (The interrupt and DMA channel that are to be used are specified in the `snd_init()` call.) Because the interrupts must be serviced frequently, the developer must ensure that interrupts do not remain off for more than 1 millisecond at a time while playing or recording sounds. The DMA channel that the sound hardware uses must not be used for any reason during the entire use of the sound library, which starts with a call to `snd_init()` and ends with a call to `snd_exit()`.

The `snd_play()` call is used to output sounds located in designated buffers. Repeated calls to `snd_play()` will cause the sounds to be played back to back, assuming that the subsequent calls are made frequently enough. The sound data provided by the caller may be expanded (depending on the sampling rate) and is placed in a series of fixed-size buffers. These buffers are placed on a queue which is handed to an internal function that programs the DMA to run continuously in the same fixed-size area. If the DMA were allowed to stop, the brief time it takes to program it to point it at the next section of the sound and start it would be audible to the listener as a "wow" or "flutter" in the sound. So the DMA runs continuously through the same buffer and the sound driver inserts subsequent portions of the sound into the buffer when interrupts and other status indicate that the DMA is transferring data in other parts of the buffer. When the DMA returns to a given point, new sound information will be present. (This "circular" DMA mode is also used in recording.)

To help synchronize sound output with other activity such as animation, `snd_wait()` allows the caller to determine when a specific sound, previously supplied to `snd_play()`, is about to start outputting. The caller can wait, then begin displaying animation or perform other tasks that need to occur at or about the same time. The `snd_flush()` call forces the player to queue any residual sound data and makes sure that the player has started. The amount of data accumulated before the player starts automatically is determined by the number of buffers provided to `snd_addbuf()`

Tandy Digital Sound Library Technical Reference

and a parameter to **snd_init()**. The **snd_wait()** function will also allow the developer to determine when the player has output all of the sounds that have been queued. The **snd_stop()** function can either terminate the sound output immediately or at a certain point. The output level can be set with **snd_volume()**, but cannot be adjusted while sound is being produced.

The **snd_record()** call causes the hardware to start digitizing sounds and stores the resulting data in an area the developer specifies in the call. The **snd_wait()** call will inform the developer when recording has finished, and the **snd_stop()** call can terminate recording at any point. A "bucket-brigade" buffering system can be implemented by making multiple calls to **snd_record()**. This allows the sound library to record into one buffer and switch to a second, while the caller is processing the first and preparing a third. This allows continuous or extended recording.

The **snd_amplify()** function is the first of a series of functions that can be used to manipulate sound data. With this call, the amplitude of a section of sound can be increased or reduced. The **snd_amplify()** call merely prepares the information needed for the manipulation. The developer calls **snd_apply_func()** to indicate what sound is to be worked on. In this way, several sounds or only portions of a given sound can be modified.

Finally, **snd_compress_part()** allows digitized sounds to be compressed so that they require less memory for storage. Some compression methods can be adjusted to be more aggressive at the cost of time and sound quality, but the compressed sound will take even less space. The compressed sounds are passed to **snd_decompress_part()**, which restores them to a playable form.

Tandy Digital Sound Libra Technical Reference

The distribution disk contains the following:

- Sound library (Including a compression and decompression algorithm which produces results identical to that used in DeskMate® Sound and Music 1988)
- Four demonstration programs, including source. One is a DeskMate application.
- Source code for some frequently needed operations.
- C include files defining structures, return codes, and other values.
- Scripts to use in building the demonstration programs.

DEMO1.EXE - Executable demo #1 which records a sound and plays it back.

DEMO1.C - main source file for demo #1.

DEMO1.H - main include declaration file for demo #1.

DEMO2.EXE - Executable demo #2 which records a sound, compresses it and stores it to disk.

DEMO2.C - main source file for demo #2.

DEMO2.H - main include declaration file for demo #2.

DEMO3.EXE - Executable demo #3 which reads a compressed sound from a file, decompresses it and plays it.

DEMO3.C - main source file for demo #3.

DEMO3.H - main include declaration file for demo #3.

DGETBUF.C - Allocates buffers for the SOUND and SNDHDR structures.

DSETUP.C - Initializes the buffers for the sound library.

DRECORD.C - Records a sound into allocated memory.

DPLAY.C - Plays back a sound from memory.

DSAVE.C - Takes a sound saved in the allocated buffers, compresses it and writes it to a file.

DLOAD.C - Decompresses a sound and loads it into the sound buffers.

SOUND.H - main include file for use with the Tandy Digital Sound Library.

DSETUP.H - Include file for initialization.

DGETBUF.H - Include file for buffer allocation.

DRECORD.H - Include file for the record function.

DPLAY.H - Include file for the playback function.

DSAVE.H - Include file for the compress and save function.

DLOAD.H - Include file for the decompression and load function.

DEMO - make file.

Tandy Digital Sound Library Technical Reference

SOUND.LIB - Tandy Digital Sound Library.

BUILD.BAT - batch file for compiling all 3 demos if make is not available.

- The disk also contains a DeskMate sub-directory which contains a demonstration program utilizing the DeskMate interface.

DMDEMO.PDM - DeskMate executable which records a sound and plays it back, compresses and stores a sound to disk, reads a compressed sound from a file, decompresses it and plays it.

DMDEMO.C - main source file for DeskMate demo.

DMGETBUF.C - DeskMate allocation of buffers for the SOUND and SNDHDR structures.

DMSETUP.C - DeskMate initialization of the buffers for the sound library.

DMRECORD.C - Records a sound into DeskMate allocated memory.

DMPLAY.C - Plays back a sound from memory utilizing DeskMate.

DMSAVE.C - Takes a sound saved in the DeskMate allocated buffers, compresses it and writes it to a file.

DMLOAD.C - Decompresses a DeskMate sound and loads it into the DeskMate sound buffers.

DMJOY.ASM - Sets and resets the Int 33 vector for the joystick.

DMDEMO.MKE - Make file for the DeskMate applications.

DEMOINFO.H - Contains all of the defines and structures for the DeskMate information boxes.

DEMODLG.H - Contains all of the defines and structures for the DeskMate dialog boxes.

All other *.H files are the same as the ones used in the non-DeskMate demos.

DMSEGS.INC - Include file for DMJOY.ASM.

DMDEMO.LNK - The link file which is called by the DeskMate make file.

DMDEMO.MAP - The map file for the DMDEMO.PDM program.

SOUND.LIB - Tandy Digital Sound Library.

Tandy Digital Sound Library Technical Reference

Structures/Defines

These type definitions are used throughout the sound library. Some are for convenience (such as the unsigned typedefs below) and some are for portability or ease of change (such as `saddr`).

```
typedef unsigned long  u_long;
typedef unsigned int   u_int;
typedef unsigned short u_short;
typedef unsigned char  u_char;

typedef char far *      saddr;

typedef struct sound {
    saddr      buffer; /* The sound buffer itself */
    u_long     sndlen; /* Length of the sound in the buffer */
    u_long     buflen; /* Length of the buffer itself */
    u_char     rate;   /* Rate the sound was recorded at */
    u_char     system; /* Sound source (see defines below) */
    u_long     reserved[4]; /* Reserved for internal use */
} SOUND;
```

Utilized by:

`snd_record()`, `snd_compress_part()`, `snd_apply_func()`, and `snd_decompress_part()` and indirectly by `snd_play()`, `snd_wait()`, and `snd_stop()`.

The `SOUND` structure is used to describe a sound or a place to put a sound when recording. A `SOUND` structure is handed to `snd_record()`, and it fills in the elements of the structure which describe the sound. The caller sets the elements `buffer` and `buflen`, and `snd_record()` fills in the rest.

`SOUND.buffer`:

A pointer to the memory where the sound itself is stored.

`SOUND.sndlen`:

The length of the sound (in bytes). It should never be longer than `buflen`.

`SOUND.buflen`:

The length of the buffer in which the sound is stored. `snd_record()` uses `buflen`.

`SOUND.rate`:

Set by `snd_record()`. It is the sampling rate at which the sound was recorded. Use the following defines:

```
/* Sampling rate defines */
#define R5500 1 /* 5.5khz sample rate */
#define R11000 2 /* 11khz sample rate */
#define R22000 3 /* 22khz sample rate */
```

`SOUND.system`:

Set by `snd_record()`. It is the type of machine on which the sound was recorded. Use the following defines:

```
/* Machine sound came from */
#define B_SYNTH 0 /* Sound was synthetically produced */
#define B_MAC 0 /* Sound came from a Macintosh(R) */
#define B_AT8 1 /* Recorded on a 8mhz AT compatible */
#define B_SL 2 /* Recorded on a 1000-SL */
#define B_TL 3 /* Recorded on a 1000-TL */
```


Tandy Digital Sound Library Technical Reference

SOUND.reserved:

Reserved for internal use. Do *not* refer to or alter this field.

```
typedef struct sndhdr {  
    /* The following items, except "ticket", "reserved1", and  
       reserved2" are to be filled in by the caller */  
    SOUND far *sndp; /* The sound structure */  
    u_long start; /* Where to start playing */  
    u_long end; /* Where to stop playing */  
    u_short ticket; /* When buffer was/is-to-be played */  
    u_char rate; /* Rate sound is to be played at */  
    u_char reserved1; /* Reserved for internal use */  
    u_long reserved2[4]; /* Also reserved for internal use */  
} SNDHDR;
```

Utilized by:

snd_play(), **snd_wait()**, and **snd_stop()**.

A SNDHDR structure is used to describe to **snd_play()** what part of a sound to play, at what rate to play it, and where the sound is. A SNDHDR points to a SOUND structure. It is important not to confuse the two structures.

SNDHDR.sndp:

The pointer to the SOUND structure, which describes the sound to be played.

SNDHDR.start:

The index into the sound from which to start playing (the playing starts at SNDHDR.sndp->buffer[start]). A value of zero (0) means to play from the start of the sound.

SNDHDR.end:

The index into the sound before which the playing is to stop (the playing ends the byte *before* SNDHDR.sndp->buffer[end]). A value of zero (0) means play to the end of the sound.

SNDHDR.rate:

The rate at which the sound is to be played back. **snd_play()** ignores the rate field in the SOUND structure.

SNDHDR.ticket:

Used internally by the sound library to identify which play buffer contains the sound.

SNDHDR.reserved1:

SNDHDR.reserved2:

Elements which are reserved for internal use. Do *not* refer to or alter these fields.

Tandy Digital Sound Library Technical Reference

```
typedef struct versioninfo {
    char far * version;
    u_char major;
    u_char minor;
    u_short edit;
    u_short reserved1;
    u_long reserved2;
    u_long reserved3;
} VERSIONINFO;

/* Version format:
   Major[Minor](Edit)*/
/*String containing version message
   that can be displayed via
   printf()*/
/*Major version number*/
/*Minor version number 0=no
   character, 1=A, 2=B, 3=C...*/
/*Edit number*/
/*Reserved for future use*/
/*Reserved for future use*/
/*Reserved for future use*/

/* Scratch area for internal use of compress/decompress routines */
typedef struct {
    u_long scratch[25]; /* Scratch area */
} COMPINFO;
```

Utilized by:

snd_compress_part() and **snd_decompress_part()**. These routines store internal information about the compression or decompression in progress in this structure. Do *not* alter or refer to anything in this structure.

```
/* Defines for compress "type" parameter */
#define CTYPE_DESKMATE88 0

/* pParam structure for compressing with type=CTYPE_DESKMATE88 */
typedef struct compparam
{
    COMPINFO far *pinfo; /* compress/decompress work area */
    u_long start; /* start position in sound */
    u_long end; /* end position in sound(0 = to the end)*/
    u_short compress_mode; /* what kind of compression to do */

    /* following parameters used only by */
    /* compress_mode = DESKMATE88_ADJUSTABLE */

    u_short threshold_length; /* minimum length of "silence" */
    u_char threshold; /* amplitude of "silence" */
    u_char precision; /* how accurate to make deltas */
} COMPPARAM;
```

COMPPARAM.pinfo:

Pointer to a COMPINFO structure. The COMPINFO structure need not be initialized to anything, it is a scratch area used by compress.

COMPPARAM.start:

The first sound sample to be included in the compression (the index to the first sample in the SOUND is zero (0).)

COMPPARAM.end:

The first sample to be *excluded* from the compression. That is, 1 plus the index of the last sample to be included in the compression. Zero (0) may be specified to indicate all bytes from start to the last byte of the sound.

Tandy Digital Sound Library Technical Reference

COMPPARAM.compress_mode:

Indicates the parameters to be used in the compression. This field should be set to one of the following, depending on the sound source:

```
/* Defines for compparam "mode" parameter */
#define DESKMATE88_MUSIC      1      /* music compression */
#define DESKMATE88_SPEECH     2      /* speech compression */
#define DESKMATE88_ADJUSTABLE 3      /* adjustable compress mode*/
```

See Appendix A for more on compression and decompression.

```
/* Compression-precision default defines */
#define SND_PRECIS            1      /*"precision" compression default */
#define SND_SPEECH_THR        12     /*"threshold" speech comp. default*/
#define SND_MUSIC_THR         0      /*"threshold" music comp. default */
#define SND_THRESH_LEN        60     /* "threshold_length" default */
```

```
/* defines for snd_cue() "mode" parameter */
#define TOPLAY                1      /* prepare to play */
#define TORECORD              2      /* prepare to record */
```

```
/* defines for snd_init() "options" parameter */
#define FASTRAMP               0x0001 /* fast change between record and
                                       play */
```

```
/* Misc defines */
#define TRUE                   1
#define FALSE                  0
```

```
#ifndef NULL
#define NULL                   0
#endif /* NULL */
```

These "defines" list all the error codes that are produced by calls in the sound library. Of the calls that do return error codes, most return at least INVALID and NOERROR. See the individual description of each call for specifics.

```
/* Error codes for sound calls */
#define BADFMT                -3     /* Decompress called with corrupted data
                                       or sound compressed with unsupported
                                       algorithm */
#define BADPARM                -2     /* Compress called with bad type or
                                       compress mode */
#define INVALID                -1     /* Call made when driver uninitialized or
                                       call not legal at this point */
#define NOERROR                0      /* Call succeeded */
#define INITBUF                1      /* Insufficient buffers were allocated
                                       before snd_init() was called */
#define INIT64K                2      /* All buffers spanned 64k addresses */
#define BADDMA                 3      /* The DMA Channel requested is invalid */
#define BADIRQ                 4      /* The IRQ number requested is invalid */
#define NOIRON                 5      /* Can't find the sound hardware */
#define BADSIZE                6      /* Record buffer is too small or too small
                                       for the requested time */
#define WOULDBLOCK             7      /* snd_record() failed because two buffers
                                       are already queued and block was FALSE
                                       */
```

Tandy Digital Sound Library Technical Reference

snd_addbuf (memory, count)
saddr memory;
u_short count;

snd_addbuf() assigns the memory pointed to by **memory** to the player's buffer pool.

Input Parameters

memory is a far pointer to the memory to be added to the player's buffer pool. If **memory** is NULL **snd_addbuf()** will return the size (in bytes) of a buffer.

count should be set to the number of buffers that will fit in the memory region pointed to by **memory**.

Return Value

NOERROR if successful.

INVALID if it is not valid to call **snd_addbuf()** at this point (i.e., **snd_init()** has been called or 128 buffers have already been added.)

Special Notes

snd_addbuf() may be called as often as desired to add buffers, until the maximum of 128 buffers have been added, or **snd_init()** is called. Any further calls to **snd_addbuf()** to add buffers will return INVALID.

To make use of the synchronizing capabilities of **snd_wait()**, the amount of buffer storage available must be greater than the size of the largest sound that could be passed to **snd_play()**. If the sounds to be played were recorded at 11 kHz, the amount of buffer space must be at least twice the size of the sound. For 5.5 kHz, at least four times the space is required.

The sound driver reserves three buffers for its internal use, and a minimum of two buffers are required for playing of sounds. Therefore, at least five buffer's worth of memory should be supplied before calling **snd_init()**. The three internal buffers are required even if **snd_play()** is not used. Also, at least one of the buffers supplied must be wholly contained in a physical 64K region (see the **snd_init()** description). This buffer is used to hold data that the DMA will read or write. The other two reserved buffers are used in certain functions for temporary storage. A minimum of two buffers (in addition to the three reserved buffers) are required to sustain continuous output.

See Also

snd_init()
snd_play()
snd_wait()
snd_exit()

Tandy Digital Sound Library Technical Reference

```
void far snd_amplify ( factor, output )  
u_short factor;  
u_char far *output;
```

snd_amplify() generates a function table to be used to amplify (or de-amplify) a sound by the specified **factor**.

Input Parameters

output is a far pointer where the table will be placed and should be defined as a 256-byte array of type `u_char`.

factor specifies the level of amplification, which is multiplied by 1000. Values for **factor** greater than 1000 amplify and values below 1000 deamplify. (For example to amplify by 1.5 times, specify 1500.)

Return Values

None.

Special Notes

The table is then used in a call to **snd_apply_func()** to actually perform the amplification.

See Also

snd_apply_func()

Tandy Digital Sound Library Technical Reference

```
void far snd_apply_func ( pSound, start, end, table )  
SOUND far *pSound;  
u_long start;  
u_long end;  
u_char far *table;
```

snd_apply_func() applies the function represented by **table** to the portion of the sound pointed to by **pSound** between **start** and **end**.

Input Parameters

pSound is a far pointer to a **SOUND** structure.

start is where to begin manipulating the sound.

end is where to stop manipulating the sound.

table is a far pointer to a 256-byte array of type **u_char**, obtained from a sound modification table generation routine, such as **snd_amplify()**.

Return Values

None.

See Also

snd_amplify()
snd_compensate()

Tandy Digital Sound Library Technical Reference

snd_compensate (adj_table)
u_char far *adj_table;

snd_compensate() generates a translation table which can be used by **snd_record()** to adjust the recorded digital data to be centered around the digital centerline (0x80).

Due to parts tolerances in the analog sound hardware (used in recording just before digitization), the digital value of absolute silence can be different from one machine to another. It can also vary over long periods of time on a single machine.

Performing this compensation is not essential, but not using compensation can cause a "click" to be output at the beginning and the end of a sound, regardless of what machine it is played on. In addition, concatenating different sounds or combining different recordings may result in distortion if the recordings were not compensated.

Input Parameters

adj_table is a far pointer to a 256-byte table which will be filled with information on whether the analog centerline is above or below the correct position. This table is then supplied to **snd_record()**, which will adjust each sample as it arrives.

Return Values

NOERROR if successful.

INVALID if it is not valid to call **snd_compensate()** at this point (i.e., **snd_init()** has not been called.)

Special Notes

To compute the centerline **snd_compensate()** reads a few thousand samples which are then averaged. While the **snd_compensate()** call is being performed, there must not be any strong sound input from the microphone or line level input. If a microphone is in use, either unplug it or get the room quiet during this time. If a line level input is present, it should be disconnected or muted during the call. If an audio signal is present while computing the centerline, the result will probably not be as accurate as it could be because most sound waveforms are not symmetrical.

snd_compensate() need only be called before the first use of record. The resulting compensation table should be correct for several hours.

See Also

snd_init()
snd_record()
snd_apply_func()

Tandy Digital Sound Library Technical Reference

```
long far snd_compress_part ( pSound, dest, dest_size, type, pParam )  
SOUND far *pSound;  
u_char far *dest;  
u_short dest_size;  
u_short type;  
COMPPARAM far *pParam;
```

snd_compress_part() performs data compression on the sound specified by **pSound**.

Input Parameters

pSound is a far pointer to a **SOUND** structure.

dest is a far pointer to a buffer of size **dest_size** characters.

type specifies the algorithm to be used to do the compression. Currently there is only one **type**, **CTYPE_DESKMATE88**.

pParam is a far pointer to a **COMPPARAM** structure where additional compression details are specified.

Return Values

Greater than zero (0) if the data compression of this segment successful. (Return value is the length of the compressed data stored into the buffer pointed to by **dest**.)

Zero (0) if the data compression of entire sound is complete.

BADPARAM if **type** is an unsupported compression algorithm

Special Notes

snd_compress_part() is called repeatedly until the entire sound has been compressed.

For each call to **snd_compress_part()** the buffer pointed to by **dest** is filled with up to **dest_size** characters, and then the number of characters stored there is returned. All calls but the first should specify **NULL** for **pSound**. On the last call, **snd_compress_part()** will return zero (0), meaning no bytes were stored in **dest**; when this happens, the compression is finished.

The contents of the **pParam** structure vary according to what **type** is set to. See Appendix A for a list of each **type** and the layout it uses for **COMPPARAM**.

See Also

Appendix A

Tandy Digital Sound Library Technical Reference

snd_cue (mode)
u_short mode;

snd_cue() puts the sound driver into standby mode for recording or playing.

Input Parameters

If **mode** is equal to **TORECORD** the driver is put into standby mode for recording.

If **mode** is equal to **TOPLAY**, the driver is put into standby mode for playing.

Return Values

NOERROR if successful.

INVALID the library is already recording or playing, or **snd_init()** has not been called.

Special Notes

When switching from play to record mode, or from record to play mode, a significant delay is encountered while the hardware is switched between modes (unless the **FASTRAMP** option has been specified to the **snd_init()** call). This delay can be over one second. In certain applications it can make cueing the start of a recording or synchronizing playback to video displays difficult, or impossible.

For example, a user could not be accurately alerted as to when recording would actually start if the delay is performed when **snd_record()** is initially called. By making the call **snd_cue(TORECORD)** before recording begins, it will perform the switch-over of the hardware, which will guarantee that recording will start immediately after calling **snd_record()**.

snd_cue() should only be called when switching from one mode (play or record) to the other. If called while the recorder or player is already running, or while the system is uninitialized, it will return an error.

A call to **snd_cue()** is not required in order to play or record. The hardware will be switched when **snd_record()** or **snd_play()** is called, if this has not already been done. However, there may be a delay before recording or playback begins.

snd_cue() does not need to be made if **snd_init()** was called with the **FASTRAMP** option.

See Also

snd_init()
snd_play()
snd_record()

Tandy Digital Sound Libra. Technical Reference

snd_decompress_part (pSound, pInfo, src, src_len)
SOUND far *pSound;
COMPINFO far *pInfo;
u_char far *src;
u_short src_len;

snd_decompress_part() decompresses a previously compressed sound.

Input Parameters

pSound is a far pointer to a SOUND structure.

pInfo is a far pointer to a COMPINFO structure.

src is a far pointer to a buffer of **src_len** length.

Return Values

NOERROR if successful.

BADFMT if the compressed sound was compressed using an unsupported compression algorithm, or is corrupted

Special Notes

snd_decompress_part() is called repeatedly until the entire sound has been decompressed.

On each call, the buffer pointed to by **src** must contain **src_len** characters; these will be decompressed and stored in the specified sound. On all but the first call, NULL should be specified for **pSound**. A last call with **src_len** set to zero (0) should be made after all data has been passed through **snd_decompress_part()**.

Before calling **snd_decompress_part()**, the caller must ensure that the buffer associated with **pSound** is at least two bytes longer than the original uncompressed sound.

The caller must fill in the length of the *uncompressed* sound in the **pSound->sndlen** field, using the length of the sound before it was compressed. The original rate and bias values from the SOUND structure at the time of compression should be placed in the new SOUND structure before playing it.

Tandy Digital Sound Library Technical Reference

snd_exit ()

snd_exit() calls **snd_stop()** if the player or recorder is running, disables the sound chip, and re-enables the joysticks. **snd_exit()** also restores the interrupt vector for the channel passed to **snd_init()** to the one that was present before **snd_init()** was called.

Input Parameters

None.

Return Values

NOERROR if successful.

INVALID if it is not valid to call **snd_exit()** at this point (i.e., **snd_init()** has not been called.)

Special Notes

snd_exit() does *not* release the memory previously supplied to **snd_addbuf()**. If a subsequent call is made to **snd_init()**, the sound library will attempt to re-use any memory that has been supplied to **snd_addbuf()** up to that point.

See Also

snd_addbuf()
snd_init()
snd_stop()

Tandy Digital Sound Libra Technical Reference

snd_flush ()

snd_flush() informs the player that there are no more sounds queued to be played in this group, and it is to immediately start playing what has been queued so far.

Input Parameters

None.

Return Values

NOERROR if successful.

INVALID if it is not valid to call **snd_flush()** at this point (i.e., **snd_play()** has not been called.)

Special Notes

If the quantity of sound queued had already reached the **trigger** threshold (see the **snd_init()** description), the player may already be running. Subsequent calls to **snd_play()** are perfectly valid, but the sound queued immediately after the **snd_flush()** will be preceded by a period of silence. The **snd_flush()** call is used to force the player to start after sounds have been queued, but before the quantity has reached the **trigger** threshold.

If the player is already running, any remaining space in the last buffer (up to 100 msec.) is filled with silence, then it is scheduled for playing. Failing to call **snd_flush()** or **snd_wait()** with the **wait** option TRUE after the last sound has been passed to **snd_play()** will result in up to 100 msec. from the end of the combined sound not being output.

When **snd_flush()** is called, any additional sound that is supplied to **snd_play()** will be separated from the earlier sounds by a period of silence. This is due to the fact that the sound driver (internal to the library) must have an exact amount of sound information in a buffer before it can be queued for playing. Calling **snd_flush()** gives the sound library permission to pad an incomplete buffer with silence and schedule it for playing. If the player runs out of sounds to play before a subsequent call to **snd_play()** can be processed, the player will go idle. The play queue will have to be filled to the **trigger** threshold or a call to **snd_flush()** or **snd_wait()** will be required to start the player again.

See Also

snd_init()
snd_play()
snd_wait()

Tandy Digital Sound Library Technical Reference

snd_init (trigger, dma_ch, irq, options)

u_short trigger;
u_short dma_ch;
u_short irq;
u_short options;

snd_init() Initializes the sound chip, disables the joysticks, sets options, and prepares to play or record.

Input Parameters

trigger is used to indicate the number of buffers allowed to accumulate before the player is forced to start.

dma_ch tells the library which DMA channel to use. Supplying a zero (0) will default to DMA channel 1.

irq indicates which interrupt channel the sound hardware is set for. See special notes.

options allows presetting various functions in the sound library. See special notes.

Return Values

NOERROR if successful.

INITBUF if insufficient buffers were added before **snd_init()** was called.

INIT64K if all buffers spanned 64k addresses.

BADDMA if the DMA channel requested is invalid.

BADIRQ if the irq number requested is invalid.

NOIRON if the sound hardware was not found.

Special Notes

A call to **snd_flush()** or to **snd_wait()** with the wait option set to TRUE will force the player to start before reaching the limit set by **trigger**. Setting **trigger** to zero (0) causes the player to fill all the supplied buffers before starting. A minimum of two buffers must accumulate before the player starts. If the **trigger** value is too low or there are not enough buffers to sustain continuous playing, the resulting sound will be broken or chopped. If this occurs, increase the number of buffers and/or set **trigger** to a higher value (or to zero).

A **trigger** value that is greater than the number of available buffers is treated as zero, ie, the player will wait until all buffers are full before starting, unless **snd_wait()** or **snd_flush()** are called first. The number of available buffers at any instant can be up to three less than the number that were supplied to **snd_addbuf()**.

Due to various hardware restrictions, only DMA channels 1 and 3 can be used for sound. On the Tandy 1000 SL and 1000 TL machines, channel 1 must be used. There is no check to confirm that the correct DMA and interrupt channel have been specified. Incorrect values will cause the sound library to malfunction.

The 1000 SL and 1000 TL machines use interrupt channel 7. Supplying a zero (0) will default to interrupt channel 7.

Tandy Digital Sound Libra., Technical Reference

Once set, the options will remain in effect until `snd_exit()` is called. They are bit fields. All undefined bits are reserved for future use and should be set to zero.

FASTRAMP Causes the switch to record from play, and from play to record to be performed instantly. When this option is selected, a "click" will be produced at the output port each time operating modes are changed. When this bit is set to zero (0), the change between operating modes is done as quietly as possible, but at the expense of time.

Once `snd_init()` is called successfully, any interrupts received on the specified channel are checked to see if they were generated by the sound hardware. If this is not the case, control is transferred to the routine that previously handled this interrupt. If this is a transient function, it must not delete the interrupt vector, mask the interrupt, or remove itself from memory without first calling `snd_exit()`.

See Also

`snd_addbuf()`
`snd_cue()`
`snd_exit()`
`snd_flush()`
`snd_play()`
`snd_record()`
`snd_wait()`

Tandy Digital Sound Library Technical Reference

snd_play (pSndHdr)
SNDHDR far *pSndHdr;

snd_play() queues the sound specified by **pSndHdr** to be played.

Input Parameters

pSndHdr is a far pointer to a **SNDHDR** structure.

Return Values

NOERROR if successful.

INVALID if it is not valid to call **snd_play()** at this point (i.e., **snd_init()** has not been called.)

Special Notes

The sound is expanded and stored in the buffers supplied to **snd_addbuf()**. The sound will not begin to play until the number of buffers accumulated thus far is equal to or greater than the **trigger** parameter supplied to **snd_init()**. **snd_play()** returns immediately unless there are no more buffers available to queue the sound in. If no buffers are available and **snd_play()** has not finished expanding the specified sound, the player is started regardless of the setting of **trigger**.

snd_play() can be called repeatedly to output a series of sounds contiguously. Sounds recorded at different sampling rates can be played together as the differences in the playback rates are handled automatically by the player.

Once the player has started, it is up to the caller to make any additional calls to **snd_play()** at a rate that is greater than the rate the previously supplied sounds are being output at to keep continuous sound output. If the caller cannot keep ahead of the player, periods of silence will be inserted between sounds, each of which should not exceed 200 msec. The amount of silence depends on the given situation; but could be infinite if the application fails to make a call to **snd_wait()**, **snd_flush()**, **snd_stop()**, or **snd_exit()** before accepting keyboard input or performing a lengthy non-sound related activity.

If the recorder is active when **snd_play()** is called, the recorder will finish filling the specified buffers before **snd_play()** processes a request. To stop the recorder immediately, call **snd_stop()**.

After all sounds in a group have been passed to **snd_play()**, a call to **snd_flush()** or to **snd_wait()** with the **wait** option equal to **TRUE** must be made, or a maximum of 100 msec. of the end of the combined sound provided to **snd_play()** will not be output. It will be played at the beginning of the next sound passed to **snd_play()**.

Tandy Digital Sound Library Technical Reference

See Also

**snd_addbuf()
snd_cue()
snd_exit()
snd_flush()
snd_init()
snd_stop()
snd_wait()**

Tandy Digital Sound Library Technical Reference

```
snd_record ( pSound, maxTime, rate, adj_table, block )  
SOUND far *pSound;  
u_short maxTime, rate;  
u_char far *adj_table;  
u_short block;
```

snd_record() places the hardware into record mode at the sampling rate specified by **rate** and fills the buffer described by **pSound** until **maxTime** milliseconds have passed or the buffer is full.

Input Parameters

pSound is a far pointer to a **SOUND** structure.

maxTime is the number in milliseconds for which the sound is to be recorded. If **maxTime** is zero (0) the buffer will be filled to the size specified by **buflen** in the **SOUND** structure. When it is non-zero, the value is converted to a number of samples to record. **buflen** is then checked to see if that number of bytes are available. If not, an error is returned.

The number of bytes per millisecond for each sampling rate is as follows:

<u>Sampling Rate</u>	<u>Bytes per millisecond</u>
22 kHz	22
11 kHz	11
5.5 kHz	5.5

Note that **snd_record()** will round **maxTime** up to the nearest number of *even* milliseconds. So, if **maxTime** is odd, and the buffer is unable to hold the extra bytes required for the additional millisecond of recording time, **snd_record()** will return **BADSIZE**.

adj_table is a far pointer to a translation table that will be used on each sample that is recorded. It is identical to the table filled in by such functions as **snd_compensate()**. The table must be filled in by calling **snd_compensate()** before recording begins. If this translation is not desired, call **snd_record()** with **adj_table** equal to "(u_char far *)NULL".

Once the recording process begins, **snd_record()** returns so that additional buffers can be prepared and **snd_record()** can be called again. If a second call to **snd_record()** is made before the first buffer has been filled, **snd_record()** will return, and the second buffer will be used after the first one is filled. If a third call is made before the first buffer is filled, **snd_record()** will wait until the recorder starts using the second buffer before returning or return with an error, depending on the **block** parameter (see below). If there is always an extra buffer available to the recorder (except at the end of recording), the group of buffers represent a continuous recording. The sampling rate specified in the first of a series of calls to **snd_record()** will be the rate used throughout the recording.

block allows the user to choose to wait until the buffer that is specified in the call can be queued, or to return immediately. (This is useful only when using more than two buffers for recording. The first two calls always return immediately.) If **block** is set to **TRUE**, **snd_record()** will wait until the active buffer is filled. Then, **snd_record()** will return. If **block** is **FALSE** and **snd_record()** already has a stand by buffer queued (ie, two calls to **snd_record()** have already been made), it will return immediately with an error. The calling program can then scan the

Tandy Digital Sound Library Technical Reference

keyboard or perform other actions, and then call **snd_record()** with the same buffer. This can be done repeatedly until the buffer is accepted.

Return Values

NOERROR if successful.

INVALID if it is not valid to call **snd_record()** at this point (i.e., **snd_init()** has not been called.)

BADSIZE if the buffer was smaller than 100 bytes or it was too small to hold the requested time.

WOULDBLOCK if two buffers are already queued and **block** was set to **FALSE**.

Special Notes

Recording buffers must be at least 100 bytes long, although less than that amount can be used by setting **maxTime** appropriately.

Recording stops when **snd_record()** has filled the buffers supplied in one or more calls, **snd_stop()** or **snd_exit()** is called, or **maxTime** is reached. After recording stops, the next call to **snd_record()** will set a new sampling rate.

As each buffer is used, the recording rate and bias fields in each sound structure are updated. This information should never be changed or discarded, otherwise the sound may play back at the wrong speed or pitch.

See Also

snd_compensate()
snd_exit()
snd_init()
snd_stop()

Tandy Digital Sound Library Technical Reference

```
void far snd_stop ( pSndHdr )  
SNDHDR far *pSndHdr;
```

snd_stop() stops the sound output or stops recording.

Input Parameters

pSndHdr is a far pointer to a SNDHDR structure.

Return Values

None.

Special Notes

While in play mode, if **pSndHdr** is NULL and the player is running, output stops within 100 msec. If **pSndHdr** is NULL and the player is not running, no sound will be output. If **pSndHdr** is not NULL, the player is not running, and the first sound in the queue is the one that is to be the stopping point, no sound is output.

Otherwise, when **pSndHdr** is not NULL, **snd_flush()** is called internally, sound output continues until the specified sound is encountered, and then output stops. If the sound is not in the play queue (it has already played or was never in the queue), output stops within 100 msec. as when **pSndHdr** is NULL. If **snd_flush()** is not called before **snd_stop()** is called and **pSndHdr** points to a sound that is within the last 100 msec. of sound passed to **snd_play()**, **snd_stop()** may not find it. If not, it will proceed as though **pSndHdr** were NULL.

After the player stops, all remaining sounds are removed from the play queue. **snd_stop()** then returns.

While in record mode, the **pSndHdr** parameter is ignored. Recording stops immediately. Up to 100 msec. of the most recently recorded sound will be discarded. The length of the sound stored in the record buffer is set, and if an additional buffer, provided by another call to **snd_record()**, was standing by, the length field in that buffer is set to zero (meaning, no contents).

See Also

snd_flush()
snd_play()

Tandy Digital Sound Library Technical Reference

VERSIONINFO far * far **snd_version** ()

snd_version() allows an application using the Tandy Digital Sound Library to determine which version of the library is being used, and to what features or options are available.

Input Parameters

None.

Return Values

A far pointer to the VERSIONINFO structure:

```
typedef struct versioninfo {  
    char far    *version;          /* String containing version  
                                   message that can be displayed  
                                   via printf() */  
    u_char      major;             /* Major version number */  
    u_char      minor;            /* Minor version number  
                                   0 = no character, 1 = A,  
                                   2 = B, 3 = C... */  
    u_short     edit;              /* Edit number */  
    u_short     reserved1;         /* Reserved for future use */  
    u_long      reserved2;         /* Reserved for future use */  
    u_long      reserved3;         /* Reserved for future use */  
} VERSIONINFO;
```

Special Notes

When writing code that determines if an option is available, use the major and minor version fields. The edit number should not be used for this purpose.

snd_version() may be called at any time, even prior to calling **snd_init()**.

Tandy Digital Sound Library Technical Reference

```
void far snd_volume ( volume )  
u_short volume;
```

snd_volume() sets the hardware gain level to **volume** for the next time the player is started.

Input Parameters

volume must be a value in the range 1 (minimum output) to 7 (maximum gain). If the player is running, the previous value for **volume** will remain in effect until the player stops.

Return Values

None.

Special Notes

The state of the player can be checked by making the call **snd_wait((SNDHDR far *)NULL,0)**. If the result is zero (0), the player is inactive, and a new gain level will be set on the next call to **snd_play()**. Calling **snd_stop()** or **snd_record()** will also guarantee that the next sound output will be at the new gain level.

The gain level defaults to 7.

snd_volume() does not affect the sound data as **snd_apply_func()** does. It merely sets the hardware output level.

See Also

```
snd_apply_func()  
snd_play()  
snd_record()  
snd_stop()  
snd_wait()
```

Tandy Digital Sound Library Technical Reference

snd_wait (pSndHdr, wait)
SNDHDR far *pSndHdr;
u_short wait;

snd_wait() provides status on the player, recorder, or a specific sound in the play queue (possibly waiting for the specified sound, the player, or the recorder to finish) and then returns.

If **pSndHdr** points to a play structure and **wait** is equal to **TRUE**, **snd_wait()** will wait until the beginning of that sound is within approximately 100 msec. of being played, and then return **TRUE**. If the sound is not on the play queue (perhaps it has already started playing), **snd_wait()** returns **TRUE** immediately.

If **pSndHdr** is **NULL** and **wait** is equal to **TRUE**, **snd_wait()** waits for *all* sounds to be completely played or all recording buffers to be filled and then returns **TRUE**.

If **pSndHdr** is a pointer to a play structure and **wait** is equal to **FALSE**, **snd_wait()** will return **TRUE** if the beginning of that sound is within approximately 100 msec. of being played, it has already started playing, or it has been played already. It will return **FALSE** otherwise.

If **pSndHdr** is **NULL** and **wait** is equal to **FALSE**, **snd_wait()** will return **TRUE** if the player or recorder is running, and **FALSE** if not.

Input Parameters

pSndHdr is a far pointer to a **SNDHDR** structure or **NULL**. See above.

wait specifies when **snd_wait()** will return. See above.

Return Values

See above.

Special Notes

The **pSndHdr** parameter is ignored while the driver is in the record mode.

If the caller queues a single (or the first) sound and then calls **snd_wait()** with **pSndHdr** pointing to the first sound and **wait** equal to **TRUE**, **snd_wait()** will call **snd_flush()** (starting the player) and return immediately since the player is always started when **wait** is **TRUE**. The second and subsequent sounds on the queue can be waited on, provided sufficient sound buffers were provided via **snd_addbuf()**. See the **snd_addbuf()** description for information on computing the appropriate number of buffers.

See Also

snd_flush()
snd_addbuf()

Tandy Digital Sound Library Technical Reference

Appendix A COMPRESSION/DECOMPRESSION

Compression is used to allow sounds to be stored in less space than they would normally require. Depending on the compression method used, significant space savings can be achieved, although some do so at the expense of sound quality.

When **snd_compress_part()** is called, the caller specifies which compression algorithm is to be applied, along with any other parameters the selected algorithm requires. The decompression call, **snd_decompress_part()**, determines which compression algorithm was used on the data and then applies the correct decompression algorithm.

The **COMPPARAM** structure is what is used to give a compression algorithm any parameters or options it needs. Thus its contents vary according to which compression algorithm is selected.

The remainder of this appendix discusses the compression types available and how to manipulate their options. The following compression algorithms are currently defined:

DESKMATE88 - Produces results identical to the compression used in DeskMate Sound Editor 1988 and the DeskMate Music 1988 application.

Tandy Digital Sound Library Technical Reference

DESKMATE 88

This compression and decompression algorithm produces results identical to the compression used in the DeskMate Sound Editor 1988 and the DeskMate Music 1988 application. To use this algorithm, the `type` parameter to `snd_compress_part()` needs to be `CTYPE_DESKMATE88` and the following fields in the `COMPPARAM` structure are filled in by the caller as follows:

`COMPPARAM.pinfo:`

Pointer to a `COMPINFO` structure. The `COMPINFO` structure need not be initialized to anything, it is a scratch area used by compress.

`COMPPARAM.start:`

The first sound sample to be included in the compression (the index to the first sample in the `SOUND` is zero (0).)

`COMPPARAM.end:`

The first sample to be *excluded* from the compression. That is, 1 plus the index of the last sample to be included in the compression. Zero (0) may be specified to indicate all bytes from start to the last byte of the sound.

`COMPPARAM.compress_mode:`

Indicates the parameters to be used in the compression. This field should be set to one of the following, depending on the sound source:

- `DESKMATE88_MUSIC` - Parameters set for music compression
- `DESKMATE88_SPEECH` - Parameters set for speech compression
- `DESKMATE88_ADJUSTABLE` - Complete control of the parameters

The caller must keep track of the length of the uncompressed sound (which can be computed by `end - start`, or `pSound->sndlen - start` if `end` is 0) and carry it with the compressed sound for use on decompression. The length of the compressed sound (the sum of the return values from `snd_compress_part()`) must also be carried with the compressed sound for use when decompressing.

This algorithm has three parameters that adjust how the compression is performed. They are fields in the `COMPPARAM` structure: `precision`, `threshold`, and `threshold_length`. These fields are *only* used when the `compress_mode` field is set to `DESKMATE88_ADJUSTABLE`. If one of the other values for `compress_mode` is selected, predefined values for `precision`, `threshold`, and `threshold_length` are assumed.

When `compress_mode` is `DESKMATE88_MUSIC`, `precision` is 1 and `threshold` is 0 (when `threshold` is 0, it doesn't matter what `threshold_length` is set to, since setting `threshold` to zero disables method C [see the discussion below]).

When `compress_mode` is `DESKMATE88_SPEECH`, `precision` is 1, `threshold` is 12, and `threshold_length` is 60.

By selecting `DESKMATE88_ADJUSTABLE` these parameters can be used to adjust the level of compression, in exchange for certain trade-offs. These trade-offs consist of more or less sound quality for less or more compression, respectively. In order to understand how to adjust the

Tandy Digital Sound Library Technical Reference

values, it will first be explained in general how the compression is done, how the variables affect the compression, and the gains and losses for each.

```
/* Compression-precision default defines */
#define SND_PRECIS 1 /* "precision" compression default */
#define SND_SPEECH_THR 12 /* "threshold" speech comp. default */
#define SND_MUSIC_THR 0 /* "threshold" music comp. default */
#define SND_THRESH_LEN 60 /* "threshold_length" default */
```

These defines are the default control parameters as available from sound.h.

The DESKMATE88 compression algorithm performs several different types of compression on the sound presented to it. These are:

- A) Scan the entire length of the sound and look to see which delta values occur more often than any others (delta values are explained below). Then, build a table of the fourteen most common values. Whenever the difference (or delta value) between the current sample and the next sample is in the table, use a four-bit quantity (sufficient to hold values from 0-13) to represent the new sample instead of the actual eight-bit sample.
- B) Replace long repetitions of the same sample value with a special code and a number telling how many times to repeat the value.
- C) Look for long repetitions of low level sound that might be pauses between words and replace this with silence (followed by an application of Method B).

Because Method B in no way affects sound quality, it has no parameters to change its behavior.

Methods A and C, however, are capable of actually removing some of the redundancy and unused or unwanted information in the sound to be compressed. Thus, parameters are available to alter their behavior. In the case of Method A, that parameter is precision. In the case of Method C those parameters are threshold and threshold_length.

Tandy Digital Sound Library Technical Reference

METHOD A

Let's say that we have the following group of sound samples from an imaginary sound (all numbers are given in hexadecimal):

80 84 86 78 74 70 89 74 80 85

On the screen of the DeskMate sound editor, sounds are displayed with 0 at the bottom of the screen, 80 at the middle (normal silence), and FF at the top of the screen. So, the data above would appear to oscillate above and below the center line displayed by the sound editor.

The difference (or delta value) between the first and second samples is 4 (84 - 80), between the third and fourth is -14 (78 - 86), and so on.

Delta values are valuable because most sounds use the entire spectrum of sample values (00 through FF) but the changes from sample to sample tend to be small and steady, thus the delta values all tend to be small and clustered around zero.

In a typical sound, fourteen delta values will account for 80% of the changes between one sample and the next. Therefore, a cache table can be built that contains the most common delta values and use only four bits to specify a cache table entry rather than eight bits to specify a new sample value.

Precision is used to decide how close a delta value needs to be to an entry in the table of most common deltas, before we will use that entry. For example, to ensure that the decompressed sound is exactly like the original, precision is set to 0. This ensures that our current difference will *exactly* match one in the table before using that table entry. However, if a close match is sufficient (and it usually is for the human ear) then it might be decided that the current difference only has to match a delta table entry within one before using the table entry. In that case, precision is set to 1.

Precision is set to 1 by the DeskMate sound editor when it compresses both speech and music. Precision values must lie between 0 and 3 with 0 being exact reproduction of the original sound and 3 being rather significant distortion.

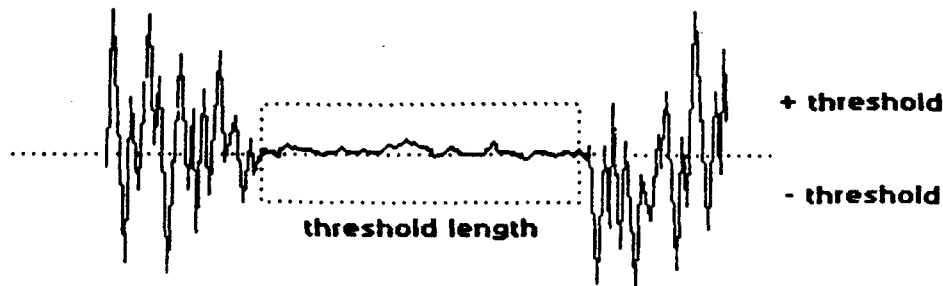
The best way to determine whether a precision of 1 or above will be acceptable on a particular sound is to try it on some similar sounds and listen to the results. A precision of 2 is fairly acceptable on speech and music with only a slight increase in "hiss" as the result.

METHOD C

Fortunately Method C is enormously easier to understand and adjust. Its two parameters specify a moving box that is centered around the "origin" or silence value of 80 (hex). This box moves forward, and if at any time the box encounters a region where *all* of the sound sample values fall within the box it assumes that this region contains only low level background noise (perhaps between spoken words) which may be replaced with silence.

Tandy Digital Sound Library Technical Reference

The two parameters determine the size of the box (and what it will consider to be background noise) in the following way:



Thus, threshold determines the height of the box and threshold_length determines the length.

Setting the value of threshold to 0 will cause compression Method C to be turned off altogether. It is recommended that this method be turned off for anything other than speech, such as recorded music or sound effects. Setting it to higher values (like 12, which is the value used for speech compression within the DeskMate sound editor) will determine how loud the background noise can get and still be removed. Setting threshold at too high a value can cause low levels of speech to be removed completely, so care must be exercised in setting this value. Setting it too low would allow small rises in the background noise level to be considered as "signal", preventing compression (removal) of this part of the noise.

The value that threshold_length is set to is multiplied by 51 in order to determine the number of samples that must fall within the threshold boundaries before being replaced with silence. Setting this value too high would cause hiss to be ignored because a long enough pause of "noise" could not be found. Setting this value too low would cause the small pauses *within* words to be replaced with silence (resulting in a strange clipped sound to the speech). The value for this parameter is 60 when speech is compressed by the DeskMate sound editor.

Tandy Digital Sound Library Technical Reference

Appendix B BIOS DIGITAL SOUND SUPPORT

The BIOS in the Tandy 1000 SL and 1000 TL computers has the same sound support sound as all previous Tandy 1000 computers and also has support for the new digital sound hardware. The interface for these new BIOS calls is defined as follows:

INT 1A

(AH) = 81H Get sound status

On return:

(AX)=00C4H

Not Busy:

(CF)=0

Busy:

(CF)=1

(AH) = 82H Input Sound (from the microphone)

On entry:

(ES:BX) = Buffer Address

(CX) = Buffer length

(DX) = Sampling Rate (8 - 4095, where 8 is the fastest supported rate)

On Return:

(AH)=00

(CF)=0 Not Busy

(CF)=1 Busy

(AH)=83H Output Sound (to the speaker)

On entry:

(ES:BX) = Buffer Address

(CX) = Buffer Length

(DX) = Sampling Rate (8 - 4095, where 8 is the fastest supported rate)

(AL) = Volume (0 - 7 where 0 = no sound)

On Return:

Same as AH = 82H

NOTE: The hardware is guaranteed to be accurate up to approximately 40 kHz (sampling rate = 8). Faster rates may produce inconsistent results from one machine to another.

(AH)=84H Stop Sound Input and Output

(AH)=85H Reinitialize joystick after any INT 1AH sound call

Tandy Digital Sound Library Technical Reference

Notes:

- The transfer rate values in register DX are not the same for calls AH=82H and AH=83H. To input a buffer of data with the AH=82H call with a given DX value and play it back with the AH=83H call so that it sounds the same, the DX value for output would have to be approximately 11.5 times as big as the DX value for input when run on a Tandy 1000 SL (8.00 MHz 8086 processor). This ratio is slightly different on other processors (8088 and 80286) and system architectures.
- These BIOS calls use the DMA hardware to input and output the sound buffer. When functions AH=82H and AH=83H are called, the BIOS initiates the I/O and returns to the calling program immediately. When the DMA transfer is complete, the BIOS receives a hardware interrupt and executes a software INT 15H with AH=91H and AL=0FBH. If an application program needs to know when the data transfer has been completed, it will have to hook INT 15H and watch for this event.
- These BIOS calls mask the hardware restriction of not being able to DMA across a 64K memory address boundary from the caller, but the sound must be less than or equal to 64K in size.

Tandy Digital Sound Library Technical Reference

Appendix C DeskMate Sound 1988 File Format

A sound file is divided into three parts: the Header, the Note Information Block, and the sound data itself.

The header is the same for all sound files and consists of the following:

Length Purpose

- | | |
|----------|--|
| Byte | X'1A' (MS-DOS end-of-file byte). |
| Byte | Compression:
0 for uncompressed file.
1 for file compressed with "music" compression (threshold equal to 0).
2 for file compressed with "speech" compression (threshold equal to 12). |
| Byte | Count. The number of "notes" defined in this file. If this is not an instrument file, this will be 1. |
| Byte | Instrument number:
0 for plain sound files.
1-32 for instrument files when this field is set.
X'FF' for instrument files that haven't had a number assigned yet. |
| 10 bytes | Instrument name. Up to 9 bytes terminated by a 0 byte. Actually, a name can be assigned even to a plain sound file, but it has no practical purpose. For an instrument file, this shows up on DeskMate's Instrument listbox in Music 1988. |
| Word | The sampling rate at which the file was originally recorded. For files recorded by the Sound Editor, this will be X'157C' (5500), X'2AF8' (11000), or X'55F0' (22000). An instrument file will always have X'2AF8' (11000). |

The Note Information Block consists of one of the following entries for each note in the file. A plain sound file will only have one of these, while an instrument file may have up to 16.

Length Purpose

- | | |
|------|---|
| Byte | Pitch. This is stored as a number from 1 (A1) to 63 (B6). If this value hasn't been set yet (for an instrument file), it will be X'FF'. |
| Byte | Reserved. |
| Byte | Low range. The lowest note that Music should use this note for. If not set, or not an instrument file, will be set to X'FF'. |
| Byte | High range. The highest note that that Music should use this note for. Set to X'FF', if not set or if plain sound file. |
| Long | Location in file. Offset from beginning of file to the actual sound data. |
| Long | Begin selection. Set to X'00000000'. Exception: in a compressed sound file, this holds the number of bytes actually used to store the sound in compressed format. |

Tandy Digital Sound Library Technical Reference

Long End selection. Set to X'00000000'.

Long End. The total number of samples in the sound. For a compressed sound, this is still the number of samples when the sound is uncompressed.

Long Begin sustain. The sample number from the start of the sound of the "sustain loop".

Long End sustain. The end of the sustain loop. If 0, the instrument has no sustain defined.

The sound data merely consists of a stream of bytes for each sound. The beginning and end of each sound is denoted only in the Note Information Block above. Note that X'80' represents a zero crossing (silence).

Tandy Digital Sound Library Technical Reference

Appendix D DeskMate 03.02.00 Environment Issues

There are several issues regarding the Tandy Digital Sound Library and the DeskMate 03.02.00 environment that the DeskMate developer must be aware of and carefully code. Otherwise the library calls may fail and cause the system to crash. An example of the correct handling of these issues is found in the DeskMate demonstration program on the distribution diskette.

Compiling

When compiling the DeskMate demo with Microsoft 'C' 5.0 the variable `__STDC__` should be defined on the compile line.

Task Switching and Accessories

The DeskMate application must observe the following rules:

1. Code shedding must be disallowed in order to protect pointers to sound buffers maintained by the sound library and other pieces of critical data.
2. When a Task Switch is recognized `snd_exit()` must be called prior to giving up control. When control is returned, `snd_init()` must then be called.
3. The memory allocated for the sound library must not be freed. The pointers to these memory areas are passed to the library via `snd_addbuf()`. The sound library assumes these buffers remain available across calls to `snd_exit()` and `snd_init()`.
4. Task Switch must not be done while compress or decompress is in progress.

Joystick

As pointed out in the Introduction and Overview section, the same DAC is used by the joysticks and for both playing and recording. Therefore, joysticks are disabled while the sound library is in use. DeskMate applications must check to see if the joystick is the current pointing device. If so, then the joystick must be disabled. Refer to the file `DMDEMO.C` in the `\DESKMATE` directory on the distribution disk for an example of correctly checking for and disabling the joystick.

There are also certain DeskMate calls which will conflict with the Sound Hardware and therefore must be avoided when the sound library is enabled (`snd_init` has been called.) They are: any of the `ms_` calls, and `event_purge()`. To work around this problem the DeskMate demo contains the two calls `Set_Joy()` and `Un_Set_Joy()` which disable the interrupt 33 vector if the joystick driver is loaded, and resets the interrupt 33 vector when finished.

Spell Resource

"Spell Resource"

Table of Contents

General Description/Notes	22- 1
Structures/Defines	22- 2
spell_bind_end	Unloads spell resource.....22- 3
spell_bind_init	Loads spell resource.....22- 4
spl_add_cache_word	Adds a word to cache ram.....22- 5
spl_add_user_word	Adds a word to the user dictionary.....22- 6
spl_check_edit	Spell checks an editfield.....22- 7
spl_check_text	Spell checks the TEXT application.....22- 8
spl_check_word	Spell checks a word.....22- 9
spl_delete_user_word ...	Deletes a word from user dictionary....22-10
spl_get_alternates	Returns similarly spelled words.....22-11
spl_init_cache	Re-initializes temporary word list.....22-12
spl_init_user_dict	Clears user dictionary memory.....22-13
spl_update_user_dict ...	Saves user dictionary to disk.....22-14

DeskMate Technical Reference Spell Check Resource

General Description/Notes

The Spell Resource (SPELL.RES) is used by the Spell Checker Accessory and by applications that wish to proof selected text. The Spell Engine Resource (SPL.RES) is the actual engine and dictionary. The engine gets loaded by SPELL.RES. SPL.RES is licensed separately from Microlytics.

The Spell Checker Resource provides two levels of spell checking to an application.

The high level calls - **spl_check_text** and **spl_check_edit** will accept a block of text as input. This block of text is parsed, presented to the user, corrections are made within the calling applications data segment. The block of text is presented to the spell checker either as a SPELL structure or an EDITFIELD structure.

The low level calls - operate upon a single word at a time. This allows applications that cannot present their data in the necessary format for the high level calls to develop their own presentation screens and user interface.

It is recommended that the user make use of the high level calls if possible.

It is necessary that application ensure that their data segment (DS) is equal to their stack segment (SS) when making calls to the Spell Checker.

DeskMate Technical Reference
Spell Check Resource
Structures/Defines

```
/*-----*/
/*
/* SPELL.H contains the Spell structure and defines */
/*
/*-----*/

struct    spell_defn
{
    char  *SPELL_start_selected;    /* pointer to selected */
    char  *SPELL_end_selected;      /* text to proof */
    char  *SPELL_text_end;          /* pointer to end of */
    char  *SPELL_end_of_buffer;     /* selected area */
    char  *SPELL_text_end;          /* pointer to end of text */
    char  *SPELL_end_of_buffer;     /* pointer to last */
    char  *SPELL_end_of_buffer;     /* position in buffer */
};
typedef struct spell_defn SPELL;

SPL_ALL_KNOWN                0
SPL_NO_ACTION                1
SPL_ACTION                   2
SPL_FULL_NO_ACTION           5
SPL_FULL_ACTION              6
SPL_DICT_TRUANT_NO_ACTION    9
SPL_DICT_TRUANT_ACTION       10
SPL_KNOWN                    0
SPL_UNKNOWN                  -1
SPL_DICT_TRUANT              -2
```

DeskMate Technical Reference

Spell Check Resource

void **spell_bind_end** ()

spell_bind_end terminates the application interface to the Spell Checker.

Input Parameters

None

Return Value

None

Special Notes

spell_bind_end should be called at the end of an application in which **spell_bind_init** was called.

DeskMate Technical Reference

Spell Check Resource

int **spell_bind_init** ()

spell_bind_init initializes a Spell Checker session.

Input Parameters

None.

Return Value

DM_ERROR if SPELL.RES or SPL.RES cannot be loaded.

NULL if successfully loaded.

Special Notes

spell_bind_init loads the Spell Checker Engine (SPL.RES), loads and initializes the user dictionary, and initializes the internal cache.

Any application which uses any of the Spell Checker functions must first initialize the Spell Checker with **spell_bind_init**.

DeskMate Technical Reference Spell Check Resource

spl_add_cache_word (pWord)
char *pWord;

spl_add_cache_word adds a word to the cache ram.

Input Parameters

pWord is a pointer to a null-terminated word.

Return Value

NULL if word added successfully.
DM_ERROR if unsuccessful.

Special Notes

The cache buffer must be initialized by calling **spl_init_cache** before words may be added.

See Also

spl_init_cache()

DeskMate Technical Reference Spell Check Resource

spl_add_user_word (pWord)
char *pWord;

spl_add_user_word adds a word to the user dictionary.

Input Parameters

pWord is a pointer to a null-terminated word

Return Value

NULL if word added successfully
DM_ERROR if unsuccessful.

Special Notes

spl_add_user_word updates the user dictionary ram. The application should ensure that changes are saved to disk by calling **spl_update_user_dict** before exiting.

See Also

spl_update_user_dict()

DeskMate Technical Reference

Spell Check Resource

```
int spl_check_edit ( pEDITFIELD )  
EDITFIELD *pEDITFIELD;
```

spl_check_edit checks the spelling of selected editfield text.

Input Parameters

pEDITFIELD is a pointer to an EDITFIELD structure

Return Value

SPL_ALL_KNOWN if all words in the selected area were recognized.

SPL_ACTION if the text was modified.

SPL_NO_ACTION if the text was **NOT** modified.

SPL_FULL_ACTION if the last spelling correction could not be made because there was not enough room in the editfield buffer for the change; buffer was previously modified.

SPL_FULL_NO_ACTION if the last spelling correction could not be made because there was not enough room in the editfield buffer for the change; buffer was **NOT** previously modified.

SPL_DICT_TRUANT_NO_ACTION if the main dictionary is no longer in the drive. Spell checking cannot continue. Changes have **NOT** been made to the text being proofed.

SPL_DICT_TRUANT_ACTION if the main dictionary is no longer in the drive. Spell checking cannot continue. Changes have been made to the text being proofed.

Special Notes

The text to be proofed should be selected.

DeskMate Technical Reference

Spell Check Resource

```
int spl_check_text ( pSPELL )  
SPELL *pSPELL;
```

spl_check_text checks the spelling of selected text in the TEXT application. If there is no selected text, the TEXT application should have the spell checker proof the whole current portion (header, footer, or document) of the current file.

Input Parameters

pSPELL is a pointer to a SPELL structure.

Return Value

SPL_ALL_KNOWN if all words in the selected area were recognized.

SPL_ACTION if the text was modified.

SPL_NO_ACTION if the text was **NOT** modified.

SPL_FULL_ACTION if the last spelling correction could not be made because there was not enough room in the TEXT buffer for the change; buffer was previously modified.

SPL_FULL_NO_ACTION if the last spelling correction could not be made because there was not enough room in the TEXT buffer for the change; buffer was **NOT** previously modified.

SPL_DICT_TRUANT_NO_ACTION if the main dictionary is no longer in the drive. Spell Checking cannot continue. Changes have **NOT** been made to the text being proofed.

SPL_DICT_TRUANT_ACTION if the main dictionary is no longer in the drive. Spell Checking cannot continue. Changes have been made to the text being proofed.

See Also

SPELL structure.

DeskMate Technical Reference

Spell Check Resource

```
int spl_check_word ( pWord )  
char  *pWord;
```

spl_check_word checks to determine if the word is in the main dictionary, user dictionary or cache storage.

Input Parameters

pWord is a pointer to a null-terminated word.

Return Value

SPL_KNOWN if the word is recognized.

SPL_UNKNOWN if the word is **NOT** known.

DeskMate Technical Reference Spell Check Resource

spl_delete_user_word (pWord)
char *pWord;

spl_delete_user_word deletes a word from the user dictionary in memory.

Input Parameters

pWord is a pointer to a null-terminated word; the word should exist in the user dictionary.

Return Value

NULL if the word was successfully deleted.

DM_ERROR if the word was not in the user dictionary.

Special Notes

After deleting words from the user dictionary, to ensure that those changes are written to disk, you must call **spl_update_user_dict**.

See Also

spl_update_user_dict()

DeskMate Technical Reference

Spell Check Resource

```
int spl_get_alternates ( pWord, pBuffer )  
char  *pWord;  
char  *pBuffer;
```

spl_get_alternates returns correctly spelled words similar in spelling to **pWord**.

Input Parameters

pWord is a pointer to a null terminated word

pBuffer is a pointer to a buffer to store the alternate words. See Special Notes below for the buffer size.

Return Value

<int>

The number of alternates found.

Special Notes

The alternate words will be separated by NULL's with the last word terminated by two NULL's.

Currently, the size of **pBuffer** should be:

$36 \text{ (maximum word length)} * 21 \text{ (maximum number of alternates)} + 1 \text{ (for null terminator)} =$
number of bytes. (757)

DeskMate Technical Reference

Spell Check Resource

int **spl_init_cache** ()

spl_init_cache tells the spell checker to re-initialize the temporary word list.

Input Parameters

None.

Return Value

None.

Special Notes

The cache provides a means to temporarily accept words for a given document that may not be appropriate for other documents, and therefore, should not be added to the user dictionary.

The cache is searched for the correct spelling of a word, but alternates are **NOT** generated from the cache.

The Spell Resource automatically initializes the ram cache upon loading via **spell_bind_init**. By calling **spl_init_cache**, the ram cache will be cleared.

DeskMate Technical Reference

Spell Check Resource

int **spl_init_user_dict** ()

spl_init_user_dict tells the spell checker to zero out the current user dictionary in memory.

Input Parameters

None.

Return Value

None.

Special Notes

The Spell Resource automatically initializes the user dictionary upon loading via **spell_bind_init**. By calling **spl_init_user_dict**, the current user dictionary in memory will be cleared.

DeskMate Technical Reference

Spell Check Resource

spl_update_user_dict ()

spl_update_user_dict displays a message box informing the user that the user dictionary has changed. The user has the option of saving the changes now or at a later time.

Input Parameters

None.

Return Value

None.

Special Notes

spl_update_user_dict should be made prior to **spell_bind_end**. **spl_update_user_dict** should not be made after every addition or deletion to the user dictionary, since it displays a message box.

If the user cancels saving the dictionary, a final message box will be displayed when the Spell Resource is removed from memory via **spell_bind_end**, which has the options to save the dictionary - OK, CANCEL. A CANCEL at this point leaves the dictionary in the state it was upon entry to the spell resource.

Thesaurus Resource

Table of Contents

General Description/Notes	23- 1
Structures/Defines	23- 3
thes_bind_endTerminates interface to the thesaurus.....	23- 4
thes_bind_initInitializes a thesaurus session.....	23- 5
thes_get_synGets synonyms for specified word.....	23- 6
thes_get_text_syn .Gets synonyms for DeskMate Text application.	23- 7

DeskMate Technical Reference Thesaurus Resource

General Description/Notes

The DeskMate Thesaurus is a resource available to applications to provide a consistent user interface with an on-line synonym dictionary. The Thesaurus resource provides the ability to locate synonyms for a word and optionally replace the word with a selected synonym to compose more readable documents.

Buffer Requirements

The word buffer should be dimensioned to 1800 bytes by the calling program. This will allow room for the largest list of synonyms now in the thesaurus.

If the chosen word is not found in the thesaurus, an attempt will be made to strip any suffix that may be present and look up the root word.

If the word is still not found, the closest alphabetical words will be in the buffer (usually about thirty words), separated by commas and concluded by a period.

If the chosen word or the root word is found in the synonym database, the buffer will contain the synonym list. The first word in the buffer will be the root word if a suffix was stripped. The synonyms are grouped by part of speech, and within part of speech by similar meaning.

The list is delimited according to the following punctuation:

: marks the beginning of the synonym list. It is placed immediately after the passed word.

, word delimiter; separates words within a group.

; group delimiter; separates synonym groups from one another.

. part of speech delimiter. Separates synonym groups based on the part of speech of the passed word.

A numeric code representing a part of speech will ALWAYS follow the colon (:) and period (.) delimiters.

The word buffer will have a structure similar to the following:

```
CHOSEN WORD:POS SYN,SYN,SYN,SYN,SYN,SYN,SYN;  
              SYN,SYN.POS SYN,SYN,SYN,SYN,SYN,SYN,SYN,  
              SYN,SYN,SYN,SYN.NULL TERMINATOR
```

where CHOSEN word is the word looked up,

POS is a Part Of Speech code,

SYN is a synonym.

The POS codes are based on the following parts of speech:

Part of Speech	POS code
noun	0
verb	1
adjective	2
adverb	3

DeskMate Technical Reference Thesaurus Resource

preposition	4
pronoun	5
transitive verb	6
connective	7
intransitive verb	8
not used	9

The chosen word, all synonyms, and the POS delimiters are stored as ASCII characters (i.e. POS code 1 = ASCII 49). The buffer is concluded with a null terminator (\0). The words in the buffer are always in lower case. The calling routine is responsible for changing the case if necessary.

Example:

Below are two samples of the synonym list which would be created from the words "flower" and "game" :

Buffer Contents: flower

flower:1advance, ascend, bloom,climb,develop,do well,excel,
expand,flourish,get ahead,grow,improve,progress,prosper,
rise,strive,survive,thrive;bloom,blossom,bud,develop,ferment,
flourish,germinate,grow,sprout.(null)

Buffer Contents: game

game:0amusement,distracton,diversion,enjoyment,entertainment,
frolic,fun,hobby,pastime,play,recreation,sport;catch,prey,quarry,
target;amusement,knickknack,plaything,toy,trinket;bout,
competition,conflict,contest,event,match,meet,meeting,race,
run.2prepared,primed,ready,ripe,set;audacious,bold,brave,
courageous,daring,dauntless,fearless,gallant,gutsy,heroic,
intrepid,stalwart,unafraid,undaunted,valiant,valorous.(null)

The example below shows the buffer contents when the chosen word was not found:

Buffer Contents: penguin

pelf,pellet,pelt,pen,pen-pusher,penalize,penalty,penchant,pending,
pendulate,pendulous,penetrate,penetrating,penetration,peninsula,
penitence,penitent,penitentiary,penmanship,pennant,penniless,
penny-pinching,pension,pensive,peon,people,pep,peppery,peppy,
perambulate,perambulatory,perceive,percentage,perceptible,
perception,perceptive,peremptory,perennial,perfect,perfected,
perfection.(null)

DeskMate Technical Reference
Thesaurus Resource
Structures/Defines

```
/*-----*/
/*
/* DMTES.H contains the Thesaurus structure and defines */
/*
/*-----*/

struct    thes_defn
{
    char *THES_cursor;          /* pointer to the cursor location */
                                /* in the buffer */
    char *THES_text_end;        /* pointer to the last character */
                                /* in the buffer */
    char *THES_buffer_end;      /* pointer to the last available */
                                /* byte in the buffer */
};
typedef    struct thes_defn THES;

THES_CANCEL      0
THES_SUBSTITUTE  1
THES_FULL        2
THES_TRUANT      3
```

DeskMate Technical Reference Thesaurus Resource

thes_bind_end ()

thes_bind_end terminates the application's interface to the thesaurus.

Input Parameters

None.

Return Value

None.

Special Notes

thes_bind_end should be called at the end of an application if **thes_bind_init** was called.

DeskMate Technical Reference Thesaurus Resource

thes_bind_init ()

thes_bind_init initializes a thesaurus session.

Input Parameters

None.

Return Value

DM_ERROR if the thesaurus resource (THES.RES) cannot be loaded.

NULL if the thesaurus resource (THES.RES) is successfully loaded.

Special Notes

Any application which wishes to use the thesaurus functions must first initialize the thesaurus with **thes_bind_init**.

DeskMate Technical Reference Thesaurus Resource

thes_get_syn (pWordBuffer)
char *pWordBuffer;

thes_get_syn returns synonyms for the word in **pWordBuffer**.

Input Parameters

pWordBuffer is a pointer to a buffer in which to store the returned synonyms. On entry, the buffer should contain the null-terminated word to lookup.

Return Value

THES_CANCEL if the user canceled, no substitutions made.

THES_SUBSTITUTE if a word was selected and should replace the current word.

THES_TRUANT if the thesaurus dictionary is not available.

Special Notes

The description of the buffer is detailed in the General Description/Notes section above.

DeskMate Technical Reference Thesaurus Resource

thes_get_text_syn (pTHES)
THES pTHES;

thes_get_text_syn provides an interface for the TEXT application with the thesaurus.
thes_get_text_syn will parse the word at the cursor position, and allow the user to replace the parsed word by selecting a substitute from a list of synonyms.

Input Parameters

pTHES is a pointer to a THES structure.

Return Value

THES_CANCEL if the user canceled, no substitutions made.
THES_SUBSTITUTE if the word was replaced.
THES_FULL if there is not enough room in the buffer to replace the word.
THES_TRUANT if the thesaurus dictionary is not available.

Titleline Manager

"Titleline Manager"

Table of Contents

General Description/Notes	24- 1
Defines	24- 2
t1l_disable_update ...Disables date/time updates.....	24- 3
t1l_enable_updateEnables date/time updates.....	24- 4
t1l_put_app_nameDisplays the application's name.....	24- 5
t1l_put_data_nameDisplays the application's data name.....	24- 6

DeskMate Technical Reference

Title Line Manager

General Description/Notes

The title line utilities manage the top line of the screen. These routines automatically display the date, time, the application name, data file, and allow the application to set an internal version number.

DeskMate Technical Reference

Title Line Manager

Defines

```
/*-----*/
/*
/* CSRBASE.H contains the Title line defines */
/*
/*-----*/

/* Title Line constants */
TTL_XORG    /* x origin of usable area of title line */
TTL_YORG    /* y origin of usable area of title line */
TTL_XEXT    /* x extent of usable area of title line */
```

DeskMate Technical Reference

Title Line Manager

tll_disable_update ()

tll_disable_update disables the date and time updates.

Input Parameters

None.

Return Value

None.

Special Notes

Once the **tll_disable_update** call is made, the date and time display will not be updated by the CSR until a call is made to the GUF resource. The GUF resource automatically enables the update of the date and time. GUF calls are also made by the critical error handler, message boxes. **tll_disable_update** will prevent the event manager from calling DOS on the **event_scan** and **event_read** services. Calls to **tll_disable_update** and **tll_enable_update** may NOT be nested.

Example

```
/* disallow the time and date to update the screen */  
tll_disable_update();
```

See Also

tll_enable_update()

DeskMate Technical Reference

Title Line Manager

ttl_enable_update ()

ttl_enable_update enables the date and time updates.

Input Parameters

None.

Return Value

None.

Special Notes

ttl_enable_update resumes the date and time display updates by the CSR after a **ttl_disable_update** call. Calls to **ttl_disable_update** and **ttl_enable_update** may NOT be nested.

Example

```
/* DeskMate may now resume displaying the */  
/* date and time updates to the title line */  
ttl_enable_update();
```

See Also

ttl_disable_update()

DeskMate Technical Reference

Title Line Manager

t1l_put_app_name (pString)
char *pString;

t1l_put_app_name displays the application's name on the title line.

Input Parameters

pString is a pointer to the application's name.

Return Value

None.

Special Notes

The application name string is limited to 9 characters, and must be terminated with a zero.

Example

```
/* display the Text Applications's name on the title line */  
t1l_put_app_name( "Text" );
```


DeskMate Technical Reference

Title Line Manager

tfl_put_data_name (pString)
char *pString;

tfl_put_data_name displays the application's current data file name on the title line.

Input Parameters

pString is a pointer to the current data file name.

Return Value

None.

Special Notes

If the name string is null, then the string (untitled) will appear on the title line.

If the name string pointer is 0L, (zero long) no name will be displayed at all.

tfl_put_data_name will automatically abbreviate the specified path if it is too long to fit on the title line.

Example

```
/* display the current data file */  
tfl_put_data_name( "TERRY.DOC" );
```

Video Manager

"Video Manager"

Table of Contents

General Description/Notes	25- 1
Structures/Defines	25- 2
csr_load_video_driver .Loads a specified video driver.....	25- 8
fvid_draw_form	Draws a form from far memory.....25- 9
fvid_draw_form_element	Draws a form element from far memory....25-10
fvid_get_bounding_box .Gets a bounding box for a far form.....	25-11
vid_busy_disable	Disables busy icon processing.....25-12
vid_busy_enable	Enables busy icon processing.....25-13
vid_clear_block	Clears specified block.....25-14
vid_clear_screen	Clears current clip region.....25-15
vid_clear_to_bot	Clears to end of clip region.....25-16
vid_clear_to_eol	Clears to end of current line.....25-17
vid_dcx_to_wcx	Converts device coordinate x.....25-18
vid_dcy_to_wcy	Converts device coordinate y.....25-19
vid_delete_line	Deletes current line.....25-20
vid_delete_nchars	Deletes specified number of characters..25-21
vid_draw_arc	Draws an arc graphic.....25-22
vid_draw_bev_rect	Draws a beveled edge rectangle.....25-23
vid_draw_busy_icon	Draws the busy icon immediately.....25-24
vid_draw_cursor	Draws the video cursor.....25-25
vid_draw_ellipse	Draws an ellipse.....25-26
vid_draw_form	Draws a form.....25-27
vid_draw_form_element .Draws a form element.....	25-28
vid_draw_frame	Draws a frame.....25-29
vid_draw_line	Draws a line.....25-30
vid_draw_point	Draws a single pixel.....25-31
vid_draw_polygon	Draws a polygon.....25-32
vid_draw_polyline	Draws a series of connected line segs...25-33
vid_draw_rect	Draws a rectangle.....25-34
vid_draw_stroke	Draws a stroke character.....25-35
vid_erase_cursor	Turns off the text cursor.....25-36
vid_fill_block	Fills a rectangle area with pattern.....25-37
vid_get_attrs	Gets the current video attributes.....25-38

vid_get_bounding_box ..Gets a bounding box for a form element..25-39
 vid_get_buffer_size ...Gets the minimum screen buffer size.....25-40
 vid_get_clipGets the current clip region.....25-42
vid_get_far_screenGets screen contents into a far buffer..25-43
 vid_get_paletteGets current RGB value for palette.....25-44
 vid_get_screenGets the screen contents into a buffer..25-45
 vid_get_viewportGets the current viewport.....25-47
 vid_get_worldGets current world boundaries.....25-48
 vid_inquire_colorsFills VID_COLOR structure.....25-49
 vid_inquire_deviceFills VID_DEVICE structure.....25-50
 vid_insert_charInserts a character onto the screen.....25-51
 vid_insert_lineInserts a line onto the screen.....25-52
 vid_invert_blockInverts fg and bg colors for a block....25-53
vid_loadable_drivers ..Returns the available video drivers.....25-54
vid_memory_freeReleases unused video memory.....25-55
vid_memory_infoReturns info about unused vid memory....25-56
vid_memory_inuseChecks for unused video memory.....25-57
 vid_move_cursorMoves cursor to specified location.....25-58
 vid_next_nwcxReturns next device pixel x.....25-59
 vid_next_nwcyReturns next device pixel y.....25-60
 vid_nextn_nwcxReturns next n device pixel x.....25-61
 vid_nextn_nwcyReturns next n device pixel y.....25-62
 vid_prev_nwcxReturns prev device pixel x.....25-63
 vid_prev_nwcyReturns prev device pixel y.....25-64
 vid_prevn_nwcxReturns prev n device pixel x.....25-65
 vid_prevn_nwcyReturns prev n device pixel y.....25-66
 vid_put_attr_nchars ...Displays attribute character pairs.....25-67
 vid_put_charDisplays character and advances cursor..25-68
vid_put_far_screenPuts far buffer contents to screen.....25-69
 vid_put_imageDisplays bitmap.....25-70
 vid_put_ncharsDisplays string of specified length.....25-72
 vid_put_screenPuts buffer contents to screen.....25-73
 vid_put_stringDisplays null terminated string.....25-74
 vid_put_ttyPuts non-filtered string to screen.....25-75
 vid_read_cursorReads current cursor location.....25-76
vid_restore_screenRestores a portion of the screen.....25-77

vid_save_screenSaves a portion of the screen.....25-78
vid_scroll_downScrolls screen down.....25-79
vid_scroll_down_no_clear ..Scrolls screen down w/o clearing ...25-80
vid_scroll_leftScrolls screen to the left.....25-81
vid_scroll_left_no_clear ..Scrolls screen left w/o clearing....25-82
vid_scroll_rightScrolls the screen to the right.....25-83
vid_scroll_right_no_clear .Scrolls screen right w/o clearing...25-84
vid_scroll_upScrolls the screen up.....25-85
vid_scroll_up_no_clearScrolls screen up w/o clearing.....25-86
vid_set_attrsFills VID_ATTRS structure.....25-87
vid_set_char_attrSets current attribute.....25-88
vid_set_char_sizeSets size of system font.....25-89
vid_set_clipSets the clip region.....25-90
vid_set_clip_to_window .Sets clip region to current window....25-91
vid_set_colorsSets fg and bg colors.....25-92
vid_set_cursor_typeSets current cursor type.....25-93
vid_set_line_attrSets current line attributes.....25-95
vid_set_paletteSets the palette colors.....25-96
vid_set_patternSets current pattern.....25-98
vid_set_viewportSets current viewport.....25-99
vid_set_worldSets current world boundaries.....25-100
vid_wcx_to_dcxConverts world coordinate x.....25-101
vid_wcx_to_nwcxConverts world coordinate x to n.....25-102
vid_wcy_to_dcyConverts world coordinate y.....25-103
vid_wcy_to_nwcxConverts world coordinate y to n.....25-104

DeskMate Technical Reference Video Manager

General Description/Notes

All direct video routines work within the current window's coordinate system and boundaries. Any parameters will be relative to the current window. All character calls will update the cursor position. Clipping will occur on any items that exceed the current window's clip boundaries. When clipping occurs, the cursor will be positioned as if the clipping did not occur.

All calls use world coordinates for device independence. The video screen measures 8000 x 5500 world coordinates. The screen is always 80 x 25 characters. A character cell measures 100 wide by 220 tall in world coordinates. Use CHAR_XEXT and CHAR_YEXT to convert from character to world coordinates.

See the Development Guide Getting Started section for a discussion of the world coordinate system.

DeskMate Technical Reference Video Manager

Structures/Defines

```
/*-----*/
/*
/* CSRVID.H contains the Video structures and defines */
/*-----*/

/* Tandy Characters */
TC_OPEN_BOX      0x01  /* box outline with a cleared interior */
TC_GRAYED_BOX    0x02  /* box outline with a grayed interior */
TC_CHECK1       0x03  /* left half of a check mark */
TC_CHECK2       0x04  /* right half of a check mark */
TC_LARROW1      0x05  /* left half of a left arrow */
TC_LARROW2      0x06  /* right half of a left arrow */
TC_RARROW1      0x06  /* left half of a right arrow */
TC_RARROW2      0x07  /* right half of a right arrow */
TC_UARROW1      0x08  /* left half of an up arrow */
TC_UARROW2      0x09  /* right half of an up arrow */
TC_DARROW1      0x0A  /* left half of a down arrow */
TC_DARROW2      0x0B  /* right half of a down arrow */
TC_ACCESSORY     0x0C  /* menu bar accessory symbol */
TC_KERNED_F1     0x0D  /* kernerd F1 character in a single character */
TC_KERNED_F      0x0E  /* left half of kernerd F1-F9 in two chars */
TC_KERNED_1      0x0F  /* right half of two character kernerd F1 */
TC_KERNED_2      0x10  /* right half of two character kernerd F2 */
TC_KERNED_3      0x11  /* right half of two character kernerd F3 */
TC_KERNED_4      0x12  /* right half of two character kernerd F4 */
TC_KERNED_5      0x13  /* right half of two character kernerd F5 */
TC_KERNED_6      0x14  /* right half of two character kernerd F6 */
TC_KERNED_7      0x15  /* right half of two character kernerd F7 */
TC_KERNED_8      0x16  /* right half of two character kernerd F8 */
TC_KERNED_9      0x17  /* right half of two character kernerd F9 */
TC_DIAMOND       0x18  /* diamond character */
TC_CATEAR1      0x19  /* left half of cat-eared page character */
TC_CATEAR2      0x1A  /* right half of cat-eared page character */
TC_KERNED_F10    0x1B  /* left half of kernerd F10 */
TC_KERNED_F102   0x1C  /* right half of kernerd F10 */
TC_BLANK        0x1F  /* alternate space character */

/* Cursor types */
VID_BLOCK_CURSOR 0      /* block cursor */
VID_BAR_CURSOR   1      /* insert cursor */
VID_LINE_CURSOR  2      /* underline cursor */
VID_DEFINED_CURSOR 3     /* definable cursor */
VID_NO_FLASH     0x80    /* mask for non-flashing cursor */

/* Character attributes */
NORMAL           0x00    /* no special attributes */
BOLD             0x01    /* bold (by color or by enbolding) */
ITALIC           0x02    /* NOT SUPPORTED italic (skewed) */
UNDERLINE        0x04    /* underlined */
INVERSE          0x08    /* inverse (swap fore & background colors) */
GRAYED           0x10    /* grayed (cross hatch shading) */
TRANSPARENT      0x20    /* transparent background */

/* Character attribute switches */
BOLD_ON          0x13
BOLD_OFF         0x12
UNDERLINE_ON     0x11
UNDERLINE_OFF    0x10
ITALIC_ON        0x17    /* NOT SUPPORTED */
ITALIC_OFF       0x16    /* NOT SUPPORTED */
```

DeskMate Technical Reference Video Manager

```
/* Colors */
COLOR_XOR      CSR_ERROR
COLOR1         0
COLOR2         1
COLOR3         2
COLOR4         3
COLOR5         4
COLOR6         5
COLOR7         6
COLOR8         7
COLOR9         8
COLOR10        9
COLOR11        10
COLOR12        11
COLOR13        12
COLOR14        13
COLOR15        14
COLOR16        15
```

```
/* Line types */
LINE_SOLID      0      /* solid lines */
LINE_INVISIBLE  1      /* invisible lines */
LINE_DOTTED     2      /* dotted lines */
LINE_DASHED     3      /* dashed lines */
LINE_DOT_DASHED 4      /* dot-dashed lines */
LINE_DENSE_DOTTED 5     /* densely dotted lines */
```

```
/* Line widths */
LINE_WIDTH1     0
LINE_WIDTH2     1
LINE_WIDTH3     2
LINE_WIDTH4     3
LINE_WIDTH5     4
```

← also: LINE_DENSE_DASHED 6 /* densely dashed lines */

```
/* Line brush */
LINE_BRUSH1     5
LINE_BRUSH2     6
LINE_BRUSH3     7
LINE_BRUSH4     8
LINE_BRUSH5     9
LINE_BRUSH6     10
LINE_BRUSH7     11
LINE_BRUSH8     12
LINE_BRUSH9     13
LINE_BRUSH10    14
LINE_BRUSH11    15
LINE_BRUSH12    16
LINE_BRUSH13    17
LINE_BRUSH14    18
LINE_BRUSH15    19
LINE_BRUSH16    20
LINE_BRUSH17    21
LINE_BRUSH18    22
```

```
/* Fill flags */
VID_FILL        SELECTED
VID_NO_FILL     DESELECTED
```

```
/* Rectangle beveledness */
VID_NO_BEVEL    0
VID_BEVEL1      1
VID_BEVEL2      2
```


DeskMate Technical Reference

Video Manager

```

VID_BEVEL3      3

/* Fill pattern types */
PATTERN_DEFAULT CSR_ERROR
PATTERN_NULL    CSR_DEFAULT    /* transparent fill */

PATTERN1        0    /* solid background */
PATTERN2        1    /* solid foreground */
PATTERN3        2    /* fine checker */
PATTERN4        3    /* coarse checker */
PATTERN5        4    /* 25% unaligned dots */
PATTERN6        5    /* 25% aligned dots */
PATTERN7        6    /* random dots */
PATTERN8        7    /* vertical stripes */
PATTERN9        8    /* horizontal stripes */
PATTERN10       9    /* ascending slanted lines */
PATTERN11      10    /* descending slanted lines */
PATTERN12      11    /* diamonds */
PATTERN13      12    /* squares */
PATTERN14      13    /* cubes */
PATTERN15      14    /* bricks */
PATTERN16      15    /* shingles */
PATTERN17      16    /* weave */
PATTERN18      17    /* epcot */
PATTERN19      18    /* curls */
PATTERN20      19    /* scales */

/* Video attributes */
struct vid_attrs_defn
{
    char    cursor_enable;    /* cursor enable flag */
    int     cursor_x;         /* cursor position */
    int     cursor_y;
    char    cursor_type;      /* block cursor type */
    int     cursor_len;       /* VID_LINE_CURSOR only */
    char    cursor_color;     /* VID_LINE_CURSOR only */
    int     cursor_defseg;    /* segment address of defined cursor */
    int     cursor_defptr;    /* offset to defined cursor */
    char    cursor_focal_x;   /* hot spot x of defined cursor */
    char    cursor_focal_y;   /* hot spot y of defined cursor */
    char    char_attr;        /* character attributes */
    char    bg_color;         /* background color */
    char    fg_color;         /* foreground color */
    char    line_type;        /* line type */
    char    line_width;       /* line width */
    char    line_fgnd_color;   /* line color */
    char    pattern;          /* fill pattern */
    int     char_xext;         /* character x extent */
    int     char_yext;         /* character y extent */
    char    line_bgnd_color;   /* line background color for brushes */
    char    overrun;          /* internal use only */
};

typedef struct vid_attrs_defn VID_ATTRS;

/* Video device */
struct vid_device_defn
{
    char    card;             /* video card id */
    int     dc_xorg;          /* device pixel x origin */
    int     dc_yorg;          /* device pixel y origin */
    int     dc_xext;          /* device pixel x extent */
    int     dc_yext;          /* device pixel y extent */
    char    xa_aspect;        /* x aspect ratio (x:y) */
    char    ya_aspect;        /* y aspect ratio (x:y) */
};

```

DeskMate Technical Reference Video Manager

```
};
typedef struct vid_device_defn VID_DEVICE;
```

VID_DEVICE.card:

Video card ID as defined below:

```
VID_1000      0      /* Tandy 1000 640x200/4 */
VID_CGA       1      /* Dual Display Adaptor/CGA 640x200/2 */
VID_DDGA      2      /* Dual Display Adaptor/special mode 640x200/16 */
VID_EGA       3      /* QuadEGA/EGA+ 640x350/16 */
VID_HERC      4      /* Dual Display Adaptor/Hercules 720x348/2 */
VID_PLAN      5      /* NOT SUPPORTED Plantronics 640x200/16 */
VID_TC16      6      /* 1988 and later Tandy 1000 custom mode 640x200/16 */
VID_TC4       7      /* NOT SUPPORTED Tandy custom mode 640x400/4 */
VID_VGA       8      /* VGA Adaptor 640x480/16 */
VID_MCGA      9      /* MCGA Adaptor */
VID_EGAM      10     /* Monochrome EGA */
VID_HERC40    14     /* Hercules 40 column 360x348/3 */
VID_EGA40C    15     /* EGA 40 column color 640x350/16 */
VID_EGA40M    16     /* EGA 40 column monochrome 640x350/3 */
VID_EGA40C_LORES 17  /* EGA 40 col color low res 320x200/16 */
```

VID_DEVICE.dc_xorg:

Device pixel x origin (-32768 to 32767).

*Video driver DMVST.RES returns 20 for VID_DEVICE.card.
Seems to be EGA w/CGA monitor (640x200x16).*

VID_DEVICE.dc_yorg:

Device pixel y origin (-32768 to 32767).

VID_DEVICE.dc_xext:

Device pixel x extent (0 to 32767).

VID_DEVICE.dc_yext:

Device pixel y extent (0 to 32767).

VID_DEVICE.xaspect:

X element of the device pixel x:y ratio (1-255).

VID_DEVICE.yaspect:

Y element of the device pixel x:y ratio (1-255).

/* Video colors */

struct vid_colors_defn

```
{
    int      nColors;          /* num of colors to choose from */
    char     nPalettes;        /* number of palettes */
    char     nGunIncs;         /* number of RGB gun increments */
    char     nIntensIncs;      /* number of intensity increments */
    char     first_palette;    /* first assignable palette of device */
};
```

typedef struct vid_colors_defn VID_COLORS;

VID_COLORS.nColors:

Number of colors to choose from for each palette (2 to 65535).

VID_COLORS.nPalettes:

Number of colors displayable on the screen at one time (2 to 255).

VID_COLORS.nGunIncs:

DeskMate Technical Reference

Video Manager

Number of increments for each of the red, green and blue color guns (1 to 64).

VID_COLORS.nIntensIncs:

Number of increments for the color intensity (1 to 255).

VID_COLORS.first_palette:

The first assignable palette of the device.

```
struct vid_palette_defn
{
    char    palette;    /* palette index */
    char    red;        /* red color gun intensity */
    char    green;      /* green color gun intensity */
    char    blue;       /* blue color gun intensity */
};
typedef struct vid_palette_defn VID_PALETTE;

/* Buttons */
struct button_defn
{
    int      xorg;       /* world coordinate x origin of button */
    int      yorg;       /* world coordinate y origin of button */
    int      xext;       /* world coordinate x extent of button */
    char     type;       /* button type */
    char     pattern;     /* pattern upon which button being drawn */
    char     bgnd_color;   /* background color of the button image */
    char     fgnd_color;   /* foreground color of the button image */
    char     string_color; /* color of the button label string */
    char     *pString;     /* pointer to the button label string */
};
typedef struct button_defn BUTTON;

/* Button masks for type element of the BUTTON structure */
BTN_ROUNDED      0x00    /* rounded edges */
BTN_SQUARED      0x01    /* squared edges */
BTN_RAISED      0x00    /* raise position */
BTN_PRESSED      0x02    /* pressed position */
BTN_INVERTED     0x04    /* invert button (valid w/ BTN_PRESSED only) */
BTN_LEFT_EDGE   0x08    /* left unobstructed (for BTN_SQUARED only) */
BTN_RIGHT_EDGE  0x10    /* right unobstructed (for BTN_SQUARED only) */
BTN_GRAYED      0x20    /* button is grayed */

/* video swapping structure */
struct vid_swap_info_defn
{
    char     driver_name[9]; /* holds the driver name */
    int      nColumns;      /* number of columns the driver supports */
    int      nColors;       /* number of colors supported */
    int      dc_xext;       /* number of pixels on screen in XEXT */
    int      dc_yext;       /* number of pixels on screen in YEXT */
    int      system_memory; /* amount of system memory in */
                                /* paragraph used */
    char     card_id;       /* holds the card id, as in the */
                                /* VID_DEVICE structure */
};
typedef struct vid_swap_info_defn VID_SWAP_INFO;

/* Video Memory */
struct vid_memory_defn
{
    char card; /* video card id */
    long start; /* offset of the first free video byte */
};
```

*changed to unsigned char
11/30/1994*

DeskMate Technical Reference

Video Manager

```
    long end ;           /* end offset of block video screen */
    long nKBytes ;       /* number of video memory Kbytes required */
};
typedef struct vid_memory_defn VID_MEMORY ;
```

DeskMate Technical Reference

Video Manager

1989 DeskMate 03.03.0x ONLY

csr_load_video_driver (pVidName)
char *pVidName;

csr_load_video_driver allows the application to swap the video driver, by loading the user requested video driver.

Input Parameters

pVidName is a pointer to a driver name string which is nine characters terminated by a NULL.

Return Value

<int>

CSR_ERROR if the specified video driver was *not* loaded.

NULL if the specified video driver was loaded.

Special Notes

If **csr_load_video_driver** is unsuccessful at loading the user specified video driver then the default driver is loaded. If the loading of the default driver is unsuccessful then **csr_load_video_driver** terminates the application through the desk executive and returns to DOS.

csr_load_video_driver should be called just after an application executes **csr_init**.

DeskMate Technical Reference Video Manager

void **evid_bind_end** (void)

evid_bind_end terminates the links to the enhanced video driver loaded by **evid_bind_init**.

Input Parameters

None.

Return Value

None.

Special Notes

evid_bind_end is only available when running DeskMate 03.05 and later.

DeskMate Technical Reference Video Manager

int **evid_bind_init** (void)

evid_bind_init loads and links to the enhanced video driver.

Input Parameters

None.

Return Value

DMF_ERROR if an error occurred in loading the resource.

Special Notes

evid_bind_init is only available when running DeskMate 03.05 and later.

DeskMate Technical Reference

Video Manager

evid_draw_form (fpFORM_HDR, Xorg, Yorg)
FORM_HDR far *fpFORM_HDR;
int Xorg;
int Yorg;

evid_draw_form draws the form pointed to by **fpFORM_HDR** with its origin determined by **Xorg**, **Yorg**.

Input Parameters

fpFORM_HDR is a far pointer to the form to be displayed.
Xorg and **Yorg** are world coordinates where the form is to be drawn.

Return Value

None.

Special Notes

evid_draw_form is only available when running DeskMate 03.03 and later.

The form should be in the format as described in the far form manager resource section of this manual. The form manager resource does not have to be loaded to call **evid_draw_form**.

Bug

The pattern attribute is destroyed when an ellipse, polyline, polygon, rectangle, or arc are rendered.

See Also

vid_draw_form()

DeskMate Technical Reference Video Manager

evid_draw_form_element (fpFORM_HDR, ElementTag)
FORM_HDR far *fpFORM_HDR;
int ElementTag;

evid_draw_form_element displays the element (or object) on the display device at its proper position on the viewport.

Input Parameters

fpFORM_HDR is a far pointer to the form to be searched.
ElementTag is the tag of the element to be displayed.

Return Value

None.

Special Notes

evid_draw_form_element is only available when running DeskMate 03.03 and later.

Bug

If the element tag does not exist the result is unpredictable. The system may lock up.

See Also

The **FORM_HDR** structure as described in the form manager section of this manual.

DeskMate Technical Reference Video Manager

evid_get_bounding_box (fpFORM_U_ARC)
FORM_U_ARC far *fpFORM_U_ARC;

evid_get_bounding_box fills in the ELEMENT structure of the FORM_U_ARC structure with the appropriate variables for the bounding box of the arc.

Input Parameters

fp_FORM_U_ARC is a far pointer to a FORM_U_ARC structure.

Return Value

None.

Special Notes

evid_get_bounding_box is only available when running DeskMate 03.03 and later.

evid_get_bounding_box fills in the x0, x0f, y0, y0f, x1, x1f, y1, and y1f parameters in the ELEMENT structure. x0, y0, x1, y1 define the bounding box of the element. **evid_get_bounding_box** is designed to be used with the FORM_U_POLYLINE and FORM_U_IMAGE structures, but currently **vid_get_bounding_box** only works for FORM_U_ARC. **evid_get_bounding_box** is normally used just prior to **eform_add_element**.

See Also

eform_add_element()

DeskMate Technical Reference

Video Manager

```
void evid_put_font ( FONT far *fpFONT )  
FONT far *fpFONT;
```

evid_put_font draws a font element using the **FONT** structure. **evid_put_font** is used by the application to draw a string of fonted characters without using the form manager. Note that **evid_put_font** uses the lower part of the same structure that would be used to place the string onto the form.

Input Parameters

fp_FONT is a far pointer to a **FONT** structure.

Return Value

None.

Special Notes

evid_put_font is only available when running DeskMate 03.05 and later.

Note that **evid_put_font** uses the lower part of the same structure that would be used to place the string onto the form.

DeskMate Technical Reference Video Manager

vid_busy_disable ()

vid_busy_disable disables busy icon processing.

Input Parameters

None.

Return Value

None.

Special Notes

After requesting vid_busy_disable, the busy icon will not appear under any circumstance.

See Also

vid_busy_enable()
vid_draw_busy_icon()

DeskMate Technical Reference

This page intentionally left blank.

DeskMate Technical Reference

Video Manager

vid_busy_enable ()

vid_busy_enable enables busy icon processing.

Input Parameters

None.

Return Value

None.

Special Notes

The busy icon defaults to the enabled state.

See Also

vid_busy_disable()
vid_draw_busy_icon()

DeskMate Technical Reference

Video Manager

vid_clear_block (Xorg, Yorg, Xext, Yext)

int Xorg;
int Yorg;
int Xext;
int Yext;

vid_clear_block clears the specified block of the screen to the current background color.

Input Parameters

The origins and extents are in world coordinates.

Return Value

None.

Special Notes

The current background color is set with **vid_set_colors**.

Example

```
/* clear rectangular block in the middle of the screen */  
vid_clear_block( 20 * CHAR_XEXT, 6 * CHAR_YEXT,  
                40 * CHAR_XEXT, 12 * CHAR_YEXT );
```

See Also

vid_fill_block()
vid_invert_block()
vid_draw_frame()
vid_set_colors()

DeskMate Technical Reference

Video Manager

vid_clear_screen ()

vid_clear_screen clears the current clipping region to the current background color. It does not home the cursor.

Input Parameters

None.

Return Value

None.

Special Notes

The current background color is set with **vid_set_colors**.

Example

```
vid_clear_screen();           /* clear the display */
vid_move_cursor( 0, 0);      /* home the cursor */
```

See Also

vid_clear_to_eol()
vid_clear_to_bot()
vid_move_cursor()
vid_set_colors()

DeskMate Technical Reference

Video Manager

vid_clear_to_bot ()

vid_clear_to_bot clears all characters from the current cursor position to the end of the current clipping region with the current background color, and also clears all rows below the current cursor position.

Input Parameters

None.

Return Value

None.

Special Notes

If the cursor position is outside of the current clipping region, the appropriate rows within the clipping region will be cleared.

Example

```
vid_move_cursor( 0, 19 * CHAR_YEXT );  
/* clear the bottom 5 lines of the screen */  
vid_clear_to_bot();
```

See Also

vid_clear_to_eol()
vid_move_cursor()
vid_delete_line()
vid_set_colors()

DeskMate Technical Reference

Video Manager

vid_clear_to_eol ()

vid_clear_to_eol clears all characters from the current cursor position to the end of the current line to the current background color.

Input Parameters

None.

Return Value

None.

Special Notes

The end of the current line is determined by the clipping region, but if the cursor is to the left of the window (or clipping region), the entire row within the clipping region will be cleared.

Example

```
vid_move_cursor( 40 * CHAR_XEXT, 11 * CHAR_YEXT );  
  
/* clear the 12th line starting at the center of the line */  
vid_clear_to_eol();
```

See Also

vid_clear_to_bot()
vid_move_cursor()
vid_delete_line()
vid_set_colors()

DeskMate Technical Reference

Video Manager

int vid_dcx_to_wcx (DeviceCordX)
int DeviceCordX;

vid_dcx_to_wcx converts a device coordinate x (relative to the screen) to a world coordinate x (relative to the active window).

Input Parameters

DeviceCordX is the device coordinate x to convert.

Return Value

<int>
A world coordinate x.

Special Notes

The world coordinate value returned will always be a normalized value.

Example

```
int    OldWorldX;  
int    NewWorldX;  
int    DeviceX;  
  
/* NewWorldX equals normalized world coordinate 100 */  
/* pixels to the right of the OldWorldX coordinate */  
DeviceX = vid_wcx_to_dcx( OldWorldX );  
DeviceX += 100;  
NewWorldX = vid_dcx_to_wcx( DeviceX );
```

See Also

vid_dcy_to_wcy()
vid_wcx_to_dcx()
vid_wcy_to_dcy()

See the Development Guide Getting Started section for a discussion of the world coordinate system.

DeskMate Technical Reference

Video Manager

```
int vid_dcy_to_wcy ( DeviceCordY )  
int DeviceCordY;
```

vid_dcy_to_wcy converts a device coordinate y (relative to the screen) to a world coordinate y (relative to the active window).

Input Parameters

DeviceCordY is the device coordinate y to convert.

Return Value

<int>
A world coordinate y.

Special Notes

The world coordinate value returned will always be a normalized value.

Example

```
int OldWorldY;  
int NewWorldY;  
int DeviceY;  
  
/* NewWorldY equals normalized world coordinate 220 */  
/* pixels down from the OldWorldY coordinate */  
DeviceY = vid_wcy_to_dcy( OldWorldY );  
DeviceY += 220;  
NewWorldY = vid_dcy_to_wcy( DeviceY );
```

See Also

vid_dcx_to_wcx()
vid_wcx_to_dcx()
vid_wcy_to_dcy()

See the Development Guide Getting Started section for a discussion of the world coordinate system.

DeskMate Technical Reference

Video Manager

vid_delete_line (OriginY)
int OriginY;

vid_delete_line deletes the character line starting at the specified world coordinate row and replaces the bottom character line with a blank character line.

Input Parameters

OriginY is the world coordinate row specifying the top of the character line to be deleted.

Return Value

None.

Example

```
/* delete the sixth line */  
vid_delete_line( 5 * CHAR_YEXT );
```

See Also

vid_insert_line()

DeskMate Technical Reference

Video Manager

vid_delete_nchars (NumChars)
int NumChars;

vid_delete_nchars deletes the specified number of characters at the current cursor position and shifts all remaining characters to the left. The characters in the last position are replaced with blanks.

Input Parameters

NumChars is the number of characters to delete. The legal range is 0 thru 132.

Return Value

None.

Special Notes

Calling **vid_delete_nchars** when the cursor is positioned outside of the current window will not shift any characters to the left.

Example

```
/* delete 5 characters */  
vid_delete_nchars( 5 );
```


DeskMate Technical Reference

Video Manager

vid_draw_bev_rect (WCX1, WCY1, WCX2, WCY2, Bevel, bFill)

int WCX1;
int WCY1;
int WCX2;
int WCY2;
char Bevel;
char bFill;

vid_draw_bev_rect draws a rectangle with beveled corners.

Input Parameters

WCX1, WCY1 one corner of the rectangle.

WCX2, WCY2 is the opposing corner.

Bevel indicates the degree of beveled edges.

VID_NO_BEVEL causes a normal rectangle to be drawn.

VID_BEVEL1 bevels the corner in 50 world coordinate units.

VID_BEVEL2 bevels the corner in 100 world coordinate units.

VID_BEVEL3 bevels the corner in 200 world coordinate units.

bFill is set to VID_FILL to fill the rectangle and VID_NO_FILL for no fill. The current pattern is used.

Return Value

None.

Special Notes

The edge uses the current line color.

The coordinates of the corners are in world coordinates.

If the rectangle is so small that the bevels will not fit, the beveled rectangle will appear as a diamond.

Example

```
/* draw a beveled rectangle 10 characters over and 10 */  
/* down, big enough to hold a 5 character word */  
vid_draw_bev_rect( 10*CHAR_XEXT, 10*CHAR_YEXT, (5*CHAR_XEXT) +  
                  PB_BASE_XEXT, PB_YEXT, VID_BEVEL1, VID_NO_FILL );
```



DeskMate Technical Reference Video Manager

vid_draw_busy_icon ()

vid_draw_busy_icon forces the busy icon to be displayed immediately.

Input Parameters

None.

Return Value

None.

Special Notes

The busy icon will be erased automatically.

See Also

vid_busy_disable()
vid_busy_enable()

DeskMate Technical Reference

Video Manager

vid_draw_cursor ()

vid_draw_cursor turns on the video cursor.

Input Parameters

None.

Return Value

None.

Special Notes

Repeated calls to **vid_draw_cursor** are not "buffered". For example, if 10 calls have been made to **vid_erase_cursor**, it only takes 1 call to **vid_draw_cursor** to turn the cursor back on.

Before a series of video calls is to be made you should first execute a **vid_erase_cursor** to eliminate cursor "flicker" and to speed up execution.

See Also

vid_erase_cursor()

DeskMate Technical Reference

Video Manager

vid_draw_ellipse (WCX1, WCY1, RadiusX, RadiusY, bFill)

int WCX1;
int WCY1;
int RadiusX;
int RadiusY;
char bFill;

vid_draw_ellipse draws an ellipse with specified vertical and horizontal radius from a center point, and fills the ellipse with the current pattern if desired.

Input Parameters

WCX1, WCY1 is the center of the ellipse.

RadiusX is the horizontal radius. i.e. the world coordinate length from the center to the left or right edge.

RadiusY is the vertical radius. i.e. the world coordinate distance from the center to the top or bottom.

bFill is set to **VID_FILL** to fill the ellipse and **VID_NO_FILL** for no fill; the current pattern is used.

Return Value

None.

Special Notes

If either **RadiusX** or **RadiusY** are zero or negative, nothing is output. If a device coordinate radius calculates to 1 then the diameter of the ellipse is 1. If the device coordinate radius is 2 then the diameter is 3. Thus diameter = (2 * radius) - 1.

Example

```
/* Center w/ 8000 x 5500 world */  
int  xRadius = 1000, yRadius = 400;  
int  xCenter = 4000, yCenter = 2250;  
  
/* draw wide ellipse */  
vid draw ellipse( xCenter, yCenter, xRadius, yRadius, VID_NO_FILL );  
/* draw a filled circle over the ellipse */  
xRadius = yRadius = 1000;  
vid_draw_ellipse( xCenter, yCenter, xRadius, yRadius, VID_FILL );
```

See Also

vid_set_pattern()

DeskMate Technical Reference

Video Manager

vid_draw_form (pFORM_HDR, Xorg, Yorg)
FORM_HDR *pFORM_HDR;
int Xorg;
int Yorg;

vid_draw_form draws the form in **pForm** with its origin determined by **xorg**, **yorg**.

Input Parameters

pFORM_HDR is a pointer to the form to be displayed.
Xorg and **Yorg** are world coordinates where the form is to be drawn.

Return Value

None.

Special Notes

The form should be in the format as described in the form manager section of this manual. The form manager resource does not have to be loaded to call **vid_draw_form**.

Bug

The pattern attribute is destroyed when an ellipse, polyline, polygon, rectangle, or arc are rendered.

See Also

fvid_draw_form()

DeskMate Technical Reference

Video Manager

vid_draw_form_element (pFORM_HDR, ElementTag)
FORM_HDR *pFORM_HDR;
int ElementTag;

vid_draw_form_element draws the element (or object) on the display device at its proper position on the viewport.

Input Parameters

pFORM_HDR is a pointer to a FORM_HDR structure.
ElementTag is the tag of the element to be displayed.

Return Value

<int>
CSR_ERROR if no such element exists.

DeskMate Technical Reference Video Manager

vid_draw_frame (pSTATIC_BOX)
STATIC_BOX *pSTATIC_BOX;

vid_draw_frame draws a frame around the specified area.

Input Parameters

pSTATIC_BOX is a pointer to a **STATIC_BOX** structure.

type is defined by the box type.

FLAT_BOX draws a single line box.

RAISED_BOX draws a "3-D" box (such as those around edit fields).

DEST_FLAT_BOX draws a **FLAT_BOX** and also erases the area that would be used by a **RAISED_BOX**.

PYRAMID draws a "3-D pyramid" box.

LISTBOX_BOX draws an item list border for list boxes.

The coordinates in **MAPRECT** for the frame are world coordinates.

color is **COLOR1** - **COLOR16**, **CSR_DEFAULT** causes the current foreground pattern to be used.

Return Value

None.

Special Notes

The background color of the frame is always the current background color. The foreground color is taken from the color value in the **STATIC_BOX** structure.

Example

```
STATIC_BOX frame;

frame.type = FLAT_BOX;
frame.maprect.xorg = 20 * CHAR_XEXT;
frame.maprect.yorg = 6 * CHAR_YEXT;
frame.maprect.xext = 40 * CHAR_XEXT;
frame.maprect.yext = 6 * CHAR_YEXT;
frame.color = COLOR3;

/* draw a frame in the center of the screen */
vid_draw_frame( frame );
```

DeskMate Technical Reference

Video Manager

vid_draw_line (WCX1, WCY1, WCX2, WCY2)

int WCX1;

int WCY1;

int WCX2;

int WCY2;

vid_draw_line draws a line from **WCX1, WCY1** to **WCX2, WCY2**. The current line color, width, and type are used.

Input Parameters

WCX1, WCY1 define one end point (world coordinates).

WCX2, WCY2, define the other end point (world coordinate).

Return Value

None.

Special Notes

Do not use a series of parallel lines to fill a region, it may not work for all device resolutions.

See Also

vid_set_line_attrs()

DeskMate Technical Reference

Video Manager

vid_draw_point (WorldCordX, WorldCordY)

int WorldCordX;

int WorldCordY;

vid_draw_point draws a single pixel at **WorldCordX**, **WorldCordY**. The current line color is used.

Input Parameters

WorldCordX and **WorldCordY** define the world coordinates of the point.

Return Value

None.

Special Notes

Do not use adjacent points to draw a short line, it will not work for all device resolutions. **vid_draw_point** always draws one pixel with color determined by the current line attributes. The width and type are ignored.

Example

```
/* draw point in the center of the screen */  
/* assuming 8000 x 5500 world */  
vid_draw_point( 4000, 2250 );
```


DeskMate Technical Reference

Video Manager

vid_draw_polygon (pPointList, NumPoints, bFill)

int *pPointList;
int NumPoints;
char bFill;

vid_draw_poly draws a polygon on the video device.

Input Parameters

pPointList is a pointer to a list of points.

NumPoints is the number of points in the list.

bFill is a flag, that when set, fills the polygon with the current pattern.

Return Value

None.

Special Notes

The format for the list of points specified by **pPointList** is wcx1, wcy1, wcx2, wcy2, wcx3, wcy3, etc. In a polygon the first and last points are always connected. If **NumPoints** is less than 2 no action is performed.

The points are relative to the active window.

See Also

vid_set_line_attrs()

DeskMate Technical Reference

Video Manager

vid_draw_polyline (pPointList, NumPointPairs)

int *pPointList;

int NumPointPairs;

vid_draw_polyline draws a series of connected line segments using the current attributes allowing the drawing of irregular shapes.

Input Parameters

pPointList is a pointer to an array of integers which represent a series of world coordinate x,y pairs.

NumPointPairs is the number of x,y pairs. If **NumPointPairs** is less than or equal to one (1) then no output is performed.

Return Value

None.

Special Notes

All line attributes are supported.

The points are relative to the active window.

Example

```
int    PolyList[8];  
  
/* initialize PolyList */  
vid_draw_polyline( PolyList, 4 );
```

See Also

vid_set_line_attrs()

DeskMate Technical Reference Video Manager

vid_draw_rect (WCX1, WCY1, WCX2, WCY2, bFill)

int WCX1;
int WCY1;
int WCX2;
int WCY2;
char bFill;

vid_draw_rect draws a rectangle using WCX1, WCY1 and WCX2, WCY2 as two opposing corners. The border is drawn using the current line attributes.

Input Parameters

WCX1, WCY1 is one corner of the rectangle **WCX2, WCY2** is the opposing corner.
If **bFill** is set to **VID_FILL** then the rectangle is filled using the current pattern. If no fill is desired use **VID_NO_FILL**.

Return Value

None.

Special Notes

vid_draw_rect uses point to point format.

See Also

See the Development Guide Getting Started section for a discussion of the world coordinate system.

DeskMate Technical Reference

Video Manager

vid_draw_stroke (pStroke, pMAPRECT, Rotate)

char *pStroke;

MAPRECT *pMAPRECT;

char Rotate;

vid_draw_stroke displays the indicated stroke character within the designated mapping rectangle.

Input Parameters

pStroke is a pointer to the stroke definition.

pMAPRECT is a pointer to a MAPRECT structure containing world coordinates giving the location to draw the stroke.

Rotate is a defined type for the attitude at which to draw the stroke: FORM_ROT0, FORM_ROT90, FORM_ROT180, FORM_ROT270.

Return Value

None.

Special Notes

The stroke font definition must be in the format described in the Forms Resource section.

DeskMate Technical Reference Video Manager

vid_erase_cursor ()

vid_erase_cursor turns off the video cursor.

Input Parameters

None.

Return Value

None.

Special Notes

Repeated calls to **vid_erase_cursor** are not "buffered". For example, if 10 calls have been made to **vid_erase_cursor**, it only takes 1 call to **vid_draw_cursor** to turn the cursor back on. Before a series of video calls is to be made you should first execute a **vid_erase_cursor** to eliminate cursor "flicker" and to speed up execution.

See Also

vid_draw_cursor()

DeskMate Technical Reference

Video Manager

vid_fill_block (Xorg, Yorg, Xext, Yext, Pattern)

int Xorg;
int Yorg;
int Xext;
int Yext;
char Pattern;

vid_fill_block fills the specified rectangular area of the screen with the specified pattern.

Input Parameters

The **origin** and **extents** are in world coordinates.

Pattern is a predefined "canned" pattern, defined by: PATTERN1 - PATTERN20.

Return Value

None.

Example

```
/* fill rectangle in the middle of the screen with random dots */  
vid_fill_block( 20 * CHAR_XEXT, 6 * CHAR_YEXT,  
                40 * CHAR_XEXT, 12 * CHAR_YEXT, PATTERN7 );
```

See Also

vid_clear_block()
vid_invert_block()
vid_draw_rect()

DeskMate Technical Reference

Video Manager

vid_get_attrs (pVID_ATTRS)
VID_ATTRS *pVID_ATTRS;

vid_get_attrs copies the current core video attributes into a VID_ATTRS structure.

Input Parameters

pVID_ATTRS is a pointer to a VID_ATTRS structure in the application's data segment.

Return Value

None.

Special Notes

vid_get_attrs is the only way to get current attributes, except the cursor position.

Example

```
VID_ATTRS    CurrentAttrs;  
vid_get_attrs( &CurrentAttrs );
```

DeskMate Technical Reference

Video Manager

vid_get_bounding_box (pFORM_U_ARC)
FORM_U_ARC *pFORM_U_ARC;

vid_get_bounding_box fills in the **ELEMENT** structure of the **FORM_U_ARC** structure with the appropriate variables for the bounding box of the arc.

Input Parameters

p_FORM_U_ARC is a pointer to a **FORM_U_ARC** structure.

Return Value

None.

Special Notes

vid_get_bounding_box fills in the **x0**, **x0f**, **y0**, **y0f**, **x1**, **x1f**, **y1**, and **y1f** parameters in the **ELEMENT** structure. **x0**, **y0**, **x1**, **y1** define the bounding box of the element. **vid_get_bounding_box** is designed to be used with the **FORM_U_POLYLINE** and **FORM_U_IMAGE** structures in addition to the **FORM_U_ARC** structure, but currently **vid_get_bounding_box** only works for **FORM_U_ARC**. **vid_get_bounding_box** is normally used just prior to **form_add_element**.

See Also

form_add_element()

DeskMate Technical Reference

Video Manager

unsigned int **vid_get_buffer_size** (Xorg, Yorg, Xext, Yext)

int Xorg;

int Yorg;

int Xext;

int Yext;

vid_get_buffer_size returns the minimum size of the buffer to use with **vid_get_screen** and **vid_put_screen**.

Input Parameters

The **origin** and **extents** are in world coordinates.

Return Value

<unsigned int>

The size of buffer in bytes.

CSR_ERROR if the buffer size is greater than or equal to 64 K.

Special Notes

Note that the value returned is device dependent. Also, the value of **xorg**, **yorg** can have a small effect on the value. This is why in some cases **xorg**, **yorg** should be normalized. Normalization assures that the value returned is only dependent on **wcxExt**, **wcyExt**. Use **vid_wcx_to_nwcx**, **vid_wcy_to_nwcy** to normalize the coordinates.

If CSR_ERROR is returned from this call the application must ask for a smaller portion of the screen to be saved, and do it's own memory management.

The table below lists the amount of memory required to execute **vid_get_screen** with extents of 1 character in each direction:

		bytes wide		scan height		number of memory planes	
CGA	=	1	x	8	x	1	= 8
1000	=	1	x	8	x	2	= 16
TC16	=	1	x	8	x	4	= 32
HERC	=	2	x	14	x	1	= 28
EGA	=	1	x	14	x	4	= 56
VGA	=	1	x	19	x	4	= 76
MCGA	=	1	x	19	x	1	= 19

Hercules characters are 9 bits wide and thus require 2 bytes per scan. The comparison is deceptive. $((100 \times 9) / 100) \times 8$

DeskMate Technical Reference

Video Manager

Example

```
int  BuffSize;
char *VidBuff;

/* get buffer size in bytes for rectangle */
BuffSize = vid_get_buffer_size( 30*CHAR_XEXT, 5*CHAR_YEXT,
                                20*CHAR_XEXT, 15*CHAR_YEXT )

if ( BuffSize == CSR_ERROR )
    ; /* reduce size of requested rectangle and call again */

if (( VidBuff = malloc( Buffsize * sizeof(char) ) ) != NULL)
    /* statements to get and/or put screen */
    .
    .
else /* not enough memory form malloc so, screen cannot be saved */
```

See Also

vid_get_screen()
vid_put_screen()
vid_wcx_to_nwcx()
vid_wcy_to_nwcy()

DeskMate Technical Reference Video Manager

vid_get_clip (pMAPRECT)
MAPRECT *pMAPRECT;

vid_get_clip gets the current clip region. The origin and extents are in world coordinates.

Input Parameters

pMAPRECT is a pointer to a MAPRECT structure in your data segment.

Return Value

The values will be returned in the structure.

Special Notes

The clip rectangle will be clipped to the device display surface. **vid_get_clip** will return values on the screen even if an attempt was made to set the clip region bigger than the screen.

Example

```
MAPRECT    ClipMR;  
vid_get_clip( &ClipMR );
```

See Also

vid_set_clip()

DeskMate Technical Reference

Video Manager

1989 DeskMate 03.03.0x ONLY

vid_get_far_screen (Xorg, Yorg, Xext, Yext)

int Xorg;

int Yorg;

int Xext;

int Yext;

vid_get_far_screen copies the bit contents of the specified rectangle on the screen into a buffer in far memory.

Input Parameters

The **origin** and **extents** of the rectangle are in world coordinates.

Return Value

The pointer to the segment of the stored screen.

CSR_ERROR if there is no memory is available.

Special Notes

The size of the buffer necessary to save a given rectangle is device dependent. The size of the buffer required can be found with **vid_get_buffer_size**. If the copy of the screen is to be moved to a different region of the screen, then **wcx, wcy** should be "normalized". Use **vid_wc_to_nwcx**, **vid_wcy_to_nwcy** to normalize the coordinates. If you are merely saving the screen to be later restored to the same location, normalization is not necessary.

Example

```
pBuffer = vid_get_far_screen( 30*CHAR_XEXT, 5*CHAR_YEXT,  
                             20*CHAR_XEXT, 15*CHAR_YEXT );
```

See Also

vid_get_buffer_size()

vid_get_screen

vid_put_screen()

vid_put_far_screen()

vid_wcx_to_nwcx()

vid_wcy_to_nwcy()

DeskMate Technical Reference

Video Manager

vid_get_palette (pVID_PALETTE)
VID_PALETTE *pVID_PALETTE;

vid_get_palette returns the RGB value for the specified palette in the R, G, and B elements in the VID_PALETTE structure.

Input Parameters

pVID_PALETTE is a pointer to a VID_PALETTE structure. The palette element specifies the palette to get.

Return Value

None.

Special Notes

Red, Green and Blue can each be 0-255 for all video drivers.

See Also

vid_set_palette()

DeskMate Technical Reference

Video Manager

vid_get_screen (Xorg, Yorg, Xext, Yext, pBuffer)

```
int Xorg;  
int Yorg;  
int Xext;  
int yext;  
char *pBuffer;
```

vid_get_screen copies the bit contents of the specified rectangle on the screen into a buffer in memory.

Input Parameters

The **origin** and **extents** of the rectangle are in world coordinates.
pBuffer is the pointer to a buffer in your data segment.

Return Value

None.

Special Notes

The size of the buffer necessary to save a given rectangle is device dependent. The size of the buffer required can be found with **vid_get_buffer_size**. If the copy of the screen is to be moved to a different region of the screen, then **wcx, wcy** should be "normalized". Use **vid_wc_to_nwcx**, **vid_wcy_to_nwcy** to normalize the coordinates. If you are merely saving the screen to be later restored to the same location, normalization is not necessary.

The table below lists the amount of memory required to execute **vid_get_screen** with extents of 1 character in each direction:

		bytes wide		scan height		number of memory planes		
CGA	=	1	x	8	x	1	=	8
1000	=	1	x	8	x	2	=	16
TC16	=	1	x	8	x	4	=	32
HERC	=	2	x	14	x	1	=	28
EGA	=	1	x	14	x	4	=	56
VGA	=	1	x	19	x	4	=	76
MCGA	=	1	x	19	x	1	=	19

Hercules characters are 9 bits wide and thus require 2 bytes per scan. The comparison is deceptive. $((100 \times 9) / 100) \times 8$

Example

```
char *VidBuff; /* pointer to allocated video buffer */  
vid_get_screen( 30*CHAR_XEXT, 5*CHAR_YEXT, 20*CHAR_XEXT,  
               15*CHAR_YEXT, VidBuff );
```

DeskMate Technical Reference

Video Manager

See Also

`vid_get_buffer_size()`
`vid_put_screen()`
`vid_wcx_to_nwcx()`
`vid_wcy_to_nwcy()`

DeskMate Technical Reference Video Manager

vid_get_viewport (pMAPRECT)
MAPRECT *pMAPRECT;

vid_get_viewport gets the current viewport. Origin and extents are in device coordinates.

Input Parameters

pMAPRECT is a pointer to a MAPRECT structure.

Return Value

None.

Special Notes

win_activate makes calls to **vid_set_world** and **vid_set_viewport** to set up your window. Normally the application programmer does not need to call **vid_set_world**, **vid_get_world**, **vid_set_viewport** or **vid_get_viewport**.

Example

```
MAPRECT    DeviceMR;

int    DeviceMR.xorg;
int    DeviceMR.yorg;
int    DeviceMR.xext;
int    DeviceMR.yext;

/* get the viewport device coordinates */
vid_get_viewport( &DeviceMR);
```

See Also

vid_set_world()
vid_get_world()
vid_set_viewport()

DeskMate Technical Reference

Video Manager

vid_get_world (pMAPRECT)
MAPRECT *pMAPRECT;

vid_get_world gets the current world boundaries by storing the world coordinates in **pMapRect**.

Input Parameters

pMAPRECT is a pointer to a MAPRECT structure in your data segment.

Return Value

None.

Special Notes

win_activate makes calls to **vid_set_world** and **vid_set_viewport** to set up your window. Normally the application programmer does not need to call **vid_set_world**, **vid_get_world**, **vid_set_viewport** or **vid_get_viewport**. The extents will be greater than zero.

Example

```
MAPRECT      WorldMR;  
  
/* get the current world boundaries */  
vid_get_world( &WorldMR );
```

See Also

win_activate()
vid_set_world()
vid_get_viewport()
vid_set_viewport()

DeskMate Technical Reference

Video Manager

vid_inquire_colors (pVID_COLORS)
VID_COLORS *pVID_COLORS;

vid_inquire_colors returns device specific color information for the currently loaded video driver.

Input Parameters

pVID_COLORS is a pointer to a VID_COLORS structure.

Return Value

None.

Special Notes

CGA can display 2 colors on the screen at a given time. Each of those colors can be selected from any of the 16 color values. For CGA, the following is returned in the VID_COLORS structure. VID_COLORS.nColors = 16, VID_COLORS.nPalettes = 2.

Example

```
VID_COLORS  Colors;  
  
/* get current video device information */  
vid_inquire_colors( &Colors );
```

See Also

vid_set_colors()
vid_set_palette()
vid_get_palette()

DeskMate Technical Reference Video Manager

vid_inquire_device (pVID_DEVICE)
VID_DEVICE *pVID_DEVICE;

vid_inquire_device returns device specific data for the currently loaded video driver.

Input Parameters

pVID_DEVICE is a pointer to a VID_DEVICE structure.

Return Value

None.

Example

```
VID_DEVICE Device;  
  
/* get the current video device information */  
vid_inquire_device( &Device );
```

Bug

In DeskMate 03.02.00 **vid_inquire_device** erroneously returned VID_EGA instead of VID_VGA. To correctly determine VGA in this situation, check the dc_yext element not equal to 350.

DeskMate Technical Reference

Video Manager

vid_insert_char (Character)
char Character;

vid_insert_char inserts a character into the current window and shifts all remaining characters in the window to the right. The character in the last column is lost.

Input Parameters

Character is the character to insert into the screen.

Return Value

None.

Special Notes

The cursor column position is always incremented, regardless of whether the cursor is outside of the window or of the physical screen. If the cursor does move outside of the current window, the cursor will continue to increment it's position, but any characters output will be clipped and no characters will be shifted to the right.

Example

```
/* inserts A at the current cursor position */  
vid_insert_char( 'A' );
```

DeskMate Technical Reference

Video Manager

vid_insert_line (Yorg)
int Yorg;

vid_insert_line inserts a blank character line starting at the specified world coordinate row. The bottom character line of the clipping region is lost.

Input Parameters

Yorg is the world coordinate row specifying the top of the character line to be inserted.

Return Value

None.

Example

```
/* insert blank line after the tenth line */  
vid_insert_line( 10 * CHAR_YEXT );
```

See Also

vid_delete_line()

DeskMate Technical Reference

Video Manager

vid_invert_block (Xorg, Yorg, Xext, Yext)

int Xorg;
int Yorg;
int Xext;
int Yext;

vid_invert_block inverts the foreground and background colors in the specified block.

Input Parameters

Xorg is the world x coordinate of the start of the block to invert.

Yorg is the world y coordinate of the start of the block to invert.

Xext is the world coordinate width of the block to invert.

Yext is the world coordinate height of the block to invert.

Return Value

None.

Special Notes

The colors are inverted as follows:

background <--> foreground

Using the default colors results in the following:

COLOR1 <--> COLOR2

COLOR3 <--> COLOR4

Changing the background and foreground colors affects which colors get swapped so that the invert will affect characters the same way the INVERSE attribute inverts character output.

Example

```
/* invert the colors in a block in the middle of the screen */
vid_invert_block( 20 * CHAR_XEXT, 6 * CHAR_YEXT,
                  40 * CHAR_XEXT, 12 * CHAR_YEXT );
```

See Also

vid_clear_block()
vid_fill_block()
vid_set_colors()

DeskMate Technical Reference

Video Manager

1989 DeskMate 03.03.0x ONLY

vid_loadable_drivers (pVID_SWAP_INFO)
VID_SWAP_INFO *pVID_SWAP_INFO;

vid_loadable_drivers determines the best video driver available which will match the current video status by using the data structure information given.

Input Parameters

pVID_SWAP_INFO is a pointer to a VID_SWAP_INFO structure which holds a list of choices available. The buffer should be the maximum video drivers that could possibly be selected (ie size of VID_SWAP_INFO * 14).

Return Value

<int>

The high byte is the number of possible drivers loadable on this card.

The low byte is the index into the buffer which is the current loaded video driver.

The VIDEO_SWAP_INFO structure is updated with the appropriate list of drivers.

DeskMate Technical Reference

Video Manager

1989 DeskMate 03.03.0x ONLY

int **vid_memory_free** ()

vid_memory_free makes it available for other applications and resources to use the memory on the video board by releasing it.

Input Parameters

None.

Return Value

NULL if successful in setting the video memory free.

CSR_ERROR if unsuccessful.

DeskMate Technical Reference

Video Manager

1989 DeskMate 03.03.0x ONLY

```
int vid_memory_info ( pVID_MEMORY )  
VID_MEMORY *pVID_MEMORY;
```

vid_memory_info initializes the values in the VID_MEMORY structure that define the beginning and end of video memory that is not in use. The only drivers that have significant video memory not in use are: EGA color and monochrome, EGA 40 column color and monochrome, EGA 40 column low resolution, and VGA.

Input Parameters

pVID_MEMORY is a pointer to a VID_MEMORY structure.

Return Value

NULL if successful in allocating video memory.

CSR_ERROR if no memory is available.

CSR_DEFAULT if the requested size is unavailable.

Special Notes

The VID_MEMORY structure is defined in CSRVID.H and CSRVID.INC

DeskMate Technical Reference

Video Manager

1989 DeskMate 03.03.0x ONLY

int **vid_memory_inuse** ()

vid_memory_inuse checks and notifies the caller if the video memory is available for use.

Input Parameters

None.

Return Value

CSR_ERROR if already in use or the memory is not available.

NULL if available to be used

DeskMate Technical Reference Video Manager

vid_move_cursor (WorldCordX, WorldCordY)

int WorldCordX;

int WorldCordY;

vid_move_cursor positions the cursor at the specified x,y world coordinates.

Input Parameters

WorldCordX is the world x coordinate to move the video cursor to.

WorldCordY is the world y coordinate to move the video cursor to.

Return Value

None.

Special Notes

The range of legal values for row and column is -32768 through 32767. There is no checking to see if the cursor has been moved off the physical screen or outside of the current window, **vid_move_cursor** is not affected by the clipping regions.

Example

```
/* upper left hand corner */
vid_move_cursor( 0,0 );
.
.
/* move cursor to position 5 characters */
/* to the right and 10 characters down */
vid_move_cursor( 5*CHAR_XEXT, 10*CHAR_YEXT );
```

DeskMate Technical Reference

Video Manager

```
int vid_next_nwcx ( WorldCordX )  
int WorldCordX;
```

vid_next_nwcx returns the normalized world coordinate x of the device pixel immediately to the right of the device pixel represented by the specified world coordinate x.

Input Parameters

WorldCordX is the world coordinate x of the device pixel to increment and convert.

Return Value

A normalized world coordinate x.

Example

```
int    CurrentWCX;  
int    NextNWCX;  
  
NextNWCX = vid_next_nwcx( CurrentWCX );
```

See Also

vid_next_nwcy()
vid_prev_nwcx()
vid_prev_nwcy()

DeskMate Technical Reference Video Manager

```
int vid_next_nwcy ( WorldCordY )  
int WorldCordY;
```

vid_next_nwcy returns the normalized world coordinate y of the device pixel immediately below the device pixel represented by the specified world coordinate y.

Input Parameters

WorldCordY is the world coordinate x of the device pixel to increment and convert.

Return Value

A normalized world coordinate y.

Example

```
int    CurrentWCY;  
int    NextNWCY;  
  
NextNWCY = vid_next_nwcy( CurrentWCY );
```

See Also

```
vid_next_nwcx()  
vid_prev_nwcx()  
vid_prev_nwcy()
```

DeskMate Technical Reference

Video Manager

```
int vid_nextn_nwcx ( WorldCordX, NumPixels )  
int WorldCordX;  
int NumPixels;
```

vid_nextn_nwcx returns the normalized world coordinate x of the device pixel n pixels to the right of the device pixel represented by the specified world coordinate x.

Input Parameters

WorldCordX is the world coordinate x of the device pixel to increment and convert.
NumPixels is the number of pixels.

Return Value

A normalized world coordinate x.

Special Notes

NumPixels can have a positive or negative value. If **NumPixels** is equal to one (1) then **vid_nextn_nwcx** performs the same function as **vid_next_nwcx**.

Example

```
int    CurrentWCX;  
int    Next5NWCX;  
  
Next5NWCX = vid_nextn_nwcx( CurrentWCX, 5 );
```

See Also

vid_nextn_nwcy()
vid_prevn_nwcx()
vid_prevn_nwcy()

DeskMate Technical Reference

Video Manager

```
int vid_nextn_nwcy ( WorldCordY, NumPixels )  
int WorldCordY;  
int NumPixels;
```

vid_nextn_nwcy returns the normalized world coordinate y of the device pixel n pixels below the device pixel represented by the specified world coordinate y.

Input Parameters

WorldCordY is the world coordinate x of the device pixel to increment and convert.
NumPixels is the number of pixels.

Return Value

A normalized world coordinate y.

Special Notes

NumPixels can have a positive or negative value. If **NumPixels** is equal to one (1) then **vid_nextn_nwcy** performs the same function as **vid_next_nwcy**.

Example

```
int    CurrentWCY;  
int    Next5NWCY;  
  
Next5NWCY = vid_nextn_nwcy( CurrentWCY, 5 );
```

See Also

vid_nextn_nwcx()
vid_prevn_nwcx()
vid_prevn_nwcy()

DeskMate Technical Reference

Video Manager

```
int vid_prev_nwcx ( WorldCordX )  
int WorldCordX;
```

vid_prev_nwcx returns the normalized world coordinate x of the device pixel immediately to the left of the device pixel represented by the specified world coordinate x.

Input Parameters

WorldCordX is the world coordinate x of the device pixel to decrement and convert.

Return Value

A normalized world coordinate x.

Example

```
int    CurrentWCX;  
int    PrevNWCX;  
  
PrevNWCX = vid_prev_nwcx( CurrentWCX );
```

See Also

```
vid_next_nwcx()  
vid_next_nwcy()  
vid_prev_nwcy()
```


DeskMate Technical Reference

Video Manager

```
int vid_prev_nwcy ( WorldCordY )  
int WorldCordY;
```

vid_prev_nwcy returns the normalized world coordinate y of the device pixel immediately above the device pixel represented by the specified world coordinate y.

Input Parameters

WorldCordY is the world coordinate y of the device pixel to decrement and convert.

Return Value

A normalized world coordinate y.

Example

```
int    CurrentWCY;  
int    PrevNWCY;  
  
PrevNWCY = vid_prev_nwcy( CurrentWCY );
```

See Also

```
vid_next_nwcx()  
vid_next_nwcy()  
vid_prev_nwcx()
```

DeskMate Technical Reference

Video Manager

```
int vid_prevn_nwcx ( WorldCordX, NumPixels )  
int WorldCordX;  
int NumPixels;
```

vid_prevn_nwcx returns the normalized world coordinate x of the device pixel n pixels to the left of the device pixel represented by the specified world coordinate x.

Input Parameters

WorldCordX is the world coordinate x of the device pixel to decrement and convert.
NumPixels is the number of pixels.

Return Value

A normalized world coordinate x.

Special Notes

NumPixels can have a positive or negative value. If **NumPixels** is equal to one (1) then **vid_prevn_nwcx** performs the same function as **vid_prev_nwcx**.

Example

```
int    CurrentWCX;  
int    Prev5NWCX;  
  
Prev5NWCX = vid_prevn_nwcx( CurrentWCX, 5 );
```

See Also

vid_nextn_nwcx()
vid_nextn_nwcy()
vid_prevn_nwcy()

DeskMate Technical Reference

Video Manager

```
int vid_prevn_nwcy ( WorldCordY, NumPixels )  
int WorldCordY;  
int NumPixels;
```

vid_prevn_nwcy returns the normalized world coordinate y of the device pixel n pixels above the device pixel represented by the specified world coordinate y.

Input Parameters

WorldCordY is the world coordinate y of the device pixel to decrement and convert.
NumPixels is the number of pixels.

Return Value

A normalized world coordinate y.

Special Notes

NumPixels can have a positive or negative value. If **NumPixels** is equal to one (1) then **vid_prevn_nwcy** performs the same function as **vid_prev_nwcy**.

Example

```
int    CurrentWCY;  
int    Prev5NWCY;  
  
Prev5NWCY = vid_prevn_nwcy( CurrentWCY, 5 );
```

See Also

```
vid_nextn_nwcy()  
vid_nextn_nwcy()  
vid_prevn_nwcy()
```

DeskMate Technical Reference

Video Manager

vid_put_attr_nchars (pString, Length)

char *pString;

int Length;

vid_put_attr_nchars accepts a pointer to a string containing attribute/character pairs, and outputs the string onto the screen at the current cursor position.

Input Parameters

pString is a pointer to a attribute/character string to put on the screen.

Length is the number of characters in the string.

Return Value

None.

Special Notes

vid_put_attr_nchars overwrites any characters that may be at the current cursor position. The attribute values are the same as those used in the **vid_set_char_attr** call.

DeskMate Technical Reference

Video Manager

vid_put_char (Character)

char Character;

vid_put_char places a character onto the screen and advances the cursor.

Input Parameters

Character is the character to place onto the screen.

Return Value

None.

Special Notes

The cursor column position is always incremented, regardless of whether the cursor is at the right edge of the window or of the physical screen. If the cursor does move past the right edge of the current window, the cursor will continue to increment it's position, but any characters output past the right edge will be clipped.

Example

```
char character = 'A';  
vid_put_char( character );  
vid_put_char( TC_ACCESSORY );
```

See Also

vid_put_nchars()

DeskMate Technical Reference

Video Manager

1989 DeskMate 03.03.0x ONLY

vid_put_far_screen (Xorg, Yorg, Xext, Yext, pBuffer)

int Xorg;
int Yorg;
int Xext;
int Yext;
char *pBuffer;

vid_put_far_screen copies the bit contents of the specified buffer into the specified rectangle on the screen.

Input Parameters

The **origin** and **extents** of the rectangle are in world coordinates.
pBuffer is the pointer to the segment of the stored screen as returned from **vid_get_far_screen**.

Return Value

None.

Special Notes

If **vid_put_far_screen** is being used to restore an area of the screen, then the parameters must be identical to those used in **vid_get_far_screen**. If the image is being moved to a different location on the screen then **wcx, wcy** should be "normalized". Use **vid_wcx_to_nwcx** and **vid_wcy_to_nwcy** to normalize the coordinates. If **vid_put_far_screen** is being used to display user defined images then **wcx, wcy** should be "normalized".

Example

```
vid_put_far_screen( 30*CHAR_XEXT, 5*CHAR_YEXT,  
                   20*CHAR_XEXT, 15*CHAR_YEXT, pBuffer );
```

See Also

vid_get_buffer_size()
vid_get_screen()
vid_get_far_screen()
vid_wcx_to_nwcx()
vid_wcy_to_nwcy()

DeskMate Technical Reference

Video Manager

vid_put_image (pIMAGE)
IMAGE *pIMAGE;

vid_put_image displays a generically defined bitmap image on the video device.

Input Parameters

pIMAGE is a pointer to an **IMAGE** structure.

Return Value

None.

Special Notes

The **IMAGE** structure is defined as follows:

```
struct image_defn
{
    MAPRECT maprect;           /* a MAPRECT structure which specifies */
                                /* the world coordinate rectangular size*/
                                /* of the displayed image. The bitmap */
                                /* definition will be expanded or */
                                /* reduced to fit this world coordinate */
                                /* area on the video device. */
    char bTransparent;         /* indicates whether the 0 color in the */
                                /* bitmap definition is to be displayed */
                                /* as color or displayed transparent. */
    int bitmap_xext;           /* specifies the pel width of the */
                                /* bitmap definition. */
    int bitmap_yext;           /* specifies the pel height of the */
                                /* bitmap definition. */
    char nColors;              /* specifies the number of colors that */
                                /* in the bitmap definition. */
    char *pBitmap;             /* a pointer to the bitmap definition. */
};
typedef struct image_defn IMAGE;
/* Number of colors in a bitmap definition */
IMAGE_2COLORS
IMAGE_4COLORS
IMAGE_16COLORS
IMAGE_256COLORS
```

The **pBitmap** definition referred to above, is a list. This list consists of numbers relating to the different colors you wish to display. To display two colors in the bitmap only one bit is used, four colors uses two bits, sixteen colors uses four bits, and 256 colors uses eight bits.

Scan lines **MUST** end on byte boundaries.

vid_put_image will expand or condense the bitmap to meet the maprect.

Example #1

to display a two color bitmap such as this:

```
-----
|color0 | color1 | color0 | color1 | color0 | color1 | color0 | color1 |
-----
|color1 | color0 | color1 | color0 | color1 | color0 | color1 | color0 |
-----
```

DeskMate Technical Reference

Video Manager

define the structure in this way:

```
MAPRECT /* where the bitmap */
/* is to be displayed */
bTransparent = DESELECTED; /* color 0 not transparent */
bitmap_xext = 8; /* 8 pixels across */
bitmap_yext = 2; /* 2 scan lines */
nColors = IMAGE 2COLORS; /* 2 colors */
*pBitmap = &SmallChecker; /* pointer to small */
/* Checkerboard definition */
/* binary definition for the small checkerboard, */
SmallChecker = 01010101, 10101010;
```

in 'C' this notation could be defined as:

```
SmallChecker = 0x55, 0xAA;
```

Example #2

to display a four color bitmap such as this:

```
-----
|color0 | color1 | color2 | color3 |
-----
|color1 | color2 | color3 | color0 |
-----
|color2 | color3 | color0 | color1 |
-----
|color3 | color0 | color1 | color2 |
-----
```

define the structure in this way:

```
MAPRECT /* where the bitmap */
/* is to be displayed */
bTransparent = DESELECTED; /* color 0 not transparent */
bitmap_xext = 4; /* 4 pixels across */
bitmap_yext = 4; /* 4 scan lines */
nColors = IMAGE 4COLORS; /* 4 colors */
*pBitmap = &BigChecker; /* pointer to big */
/* Checkerboard definition */
/* binary definition for the big checkerboard */
BigChecker = 00011011, 01101100, 10110001, 11000110;
```

in 'C' this notation could be defined as:

```
BigChecker = 0x1B, 0x6C, 0xB1, 0xC6;
```

Note: in these examples color0 = 0, color1 = 1, color2 = 2 etc. They are not to be construed to be equal to the defines in CSRVID.H of COLOR1, COLOR2 etc.

DeskMate Technical Reference

Video Manager

vid_put_nchars (pString, Length)

char *pString;

int Length;

vid_put_nchars outputs a string of the specified length onto the screen.

Input Parameters

pString is a pointer to the string to output to the screen.

Length is the number of characters in the string.

Return Value

None.

Special Notes

vid_put_nchars overwrites any text that may currently be at the current cursor position. The string will be clipped on both the left and right edges of the current window. The cursor is always incremented to the position following the end of the string, regardless of whether it was clipped or not. The maximum length of the a string that can be used is 132 characters.

Example

```
char *string = "Hello World";
```

```
/* print out Hello */  
vid_put_nchars( string, 5 );
```

DeskMate Technical Reference

Video Manager

vid_put_screen (Xorg, Yorg, Xext, Yext, pBuffer)

```
int Xorg;  
int Yorg;  
int Xext;  
int Yext;  
char *pBuffer
```

vid_put_screen copies the bit contents of the specified buffer into the specified rectangle on the screen.

Input Parameters

The **origin** and **extents** of the rectangle are in world coordinates.
pBuffer is the pointer to a buffer in your data segment.

Return Value

None.

Special Notes

If **vid_put_screen** is being used to restore an area of the screen, then the parameters must be identical to those used in the **vid_get_screen** call. If the image is being moved to a different location on the screen then **wcx, wcy** should be "normalized". Use **vid_wcx_to_nwcx** and **vid_wcy_to_nwcy** to normalize the coordinates. If **vid_put_screen** is being used to display user defined images then **wcx, wcy** should be "normalized".

Example

```
char *VidBuff    /* pointer to video buffer with screen data */  
vid_put_screen( 30*CHAR_XEXT, 5*CHAR_YEXT, 20*CHAR_XEXT,  
                15*CHAR_YEXT, VidBuff );
```

See Also

```
vid_get_buffer_size()  
vid_get_screen()  
vid_wcx_to_nwcx()  
vid_wcy_to_nwcy()
```

DeskMate Technical Reference

Video Manager

vid_put_string (pString)
char *pString;

vid_put_string outputs a string onto the screen at the current cursor position and moves the cursor to the position following the string.

Input Parameters

pString is a pointer to a null terminated character string.

Return Value

None.

Special Notes

vid_put_string overwrites any text that may currently be at the current cursor position. The string will be clipped on both the left and right edges of the current window. The cursor is always incremented to the position following the end of the string, regardless of whether it was clipped or not. The maximum length of the a string that can be used is 132 characters plus a null terminator.

Example

```
char *String = "Hello World!";  
vid_put_string( String );  
vid_put_string( "Hello Again." );
```

DeskMate Technical Reference

Video Manager

vid_put_tty (Character)
char Character;

vid_put_tty outputs a character to the screen and wraps the cursor at the right edge of the screen and scrolls to the bottom edge of the screen.

Input Parameters

Character is the character to output to the screen.

Return Value

None.

Special Notes

The following TTY control characters are supported:

RETURN_KEY (0x0D) performs a carriage return.

LINE_FEED (0x0A) performs a line feed.

BKSPACE performs a destructive backspace.

Example

```
char character = 'A';  
vid_put_tty( character );      /* prints A on the screen */  
vid_put_tty( BKSPACE );      /* deletes the A just displayed */
```

DeskMate Technical Reference Video Manager

vid_read_cursor (pWcx, pWcy)

int *pWcx;

int *pWcy;

vid_read_cursor returns the current x,y world coordinates of the video cursor.

Input Parameters

pWcx is a pointer to the variable in which to place the cursor's x coordinate.

pWcy is a pointer to the variable in which to place the cursor's y coordinate.

Return Value

None.

Special Notes

The range of legal column and row coordinates is -32768 through 32767.

Example

```
int  CursorX;  
int  CursorY;
```

```
vid_read_cursor( &CursorX, &CursorY );
```

See Also

vid_move_cursor()

DeskMate Technical Reference

Video Manager

vid_restore_screen ()

vid_restore_screen restores the region saved by the most recent **vid_save_screen** call. It does not matter what window you are in when calling **vid_restore_screen**.

Input Parameters

None.

Return Value

None.

Special Notes

vid_restore_screen is only available when running DeskMate 03.03 and later.

vid_save_screen and **vid_restore_screen** use an internal buffer that has already been allocated by the CSR. This internal buffer is used to save the screen behind message boxes and menus. Thus, if **vid_save_screen** is called be sure that message boxes and menus are not used until after **vid_restore_screen** is called. If you require your "own" buffer, see **vid_get_far_screen** and **vid_put_far_screen**.

See Also

vid_save_screen()

DeskMate Technical Reference

Video Manager

vid_save_screen (WcxOrg, WcyOrg, WcxExt, WcyExt)

int WcxOrg;

int WcyOrg;

int WcxExt;

int WcyExt;

vid_save_screen saves the specified region relative to the current window.

Input Parameters

WcxOrg and **WcyOrg** are the x and y origin from which to start the screen save.

WcxExt and **WcyExt** are the x and y extent to which the screen is saved.

Return Value

None.

Special Notes

vid_save_screen is only available when running DeskMate 03.03 and later.

The maximum amount of screen that can be saved using **vid_save_screen** is 4200 x 1980 world coordinates. This corresponds to 42 * CHAR_XEXT and 9 * CHAR_YEXT or 378 characters. This is the same size as a message box when using the 80 column video drivers. There will be some variance if the characters are not on character boundaries.

vid_save_screen and **vid_restore_screen** use an internal buffer that has already been allocated by the CSR. This internal buffer is used to save the screen behind message boxes and menus. Thus, if **vid_save_screen** is called be sure that message boxes and menus are not used until after **vid_restore_screen** is called. If you require your "own" buffer, see **vid_get_far_screen** and **vid_put_far_screen**."

DeskMate Technical Reference

Video Manager

vid_scroll_down (Yext)
int Yext;

vid_scroll_down scrolls the screen within the current clipping region down the specified number of world coordinate units. The top part of the clipping region is cleared to the current background color.

Input Parameters

Yext is the number of world coordinate units to scroll the screen down.

Return Value

None.

Special Notes

If **Yext** is less than or equal to zero, no action is performed.

Example

```
/* scroll down one line */  
vid_scroll_down( CHAR_YEXT );
```

See Also

vid_scroll_up()
vid_scroll_right()
vid_scroll_left()
vid_scroll_up_no_clear()
vid_scroll_down_no_clear()
vid_scroll_left_no_clear()
vid_scroll_right_no_clear()

DeskMate Technical Reference Video Manager

vid_scroll_down_no_clear (Yext)

int Yext;

vid_scroll_down_no_clear scrolls the screen within the current clipping region down the specified number of world coordinate units. The top of the clipping region is not cleared.

Input Parameters

Yext is the number of world coordinate units to scroll the screen down.

Return Value

None.

Special Notes

If **Yext** is less than or equal to zero, no action is performed.

See Also

vid_scroll_up()
vid_scroll_down()
vid_scroll_left()
vid_scroll_right()
vid_scroll_up_no_clear()
vid_scroll_left_no_clear()
vid_scroll_right_no_clear()

DeskMate Technical Reference

Video Manager

vid_scroll_left (Xext)

int Xext;

vid_scroll_left scrolls the screen within the current clipping region left the specified number of world coordinate units. The right part of the clipping region is cleared to the current background color.

Input Parameters

Xext is the number of world coordinate units to scroll the screen left.

Return Value

None.

Special Notes

If **Xext** is less than or equal to zero, no action is performed.

Example

```
/* scroll left on character */  
vid_scroll_left( CHAR_XEXT );
```

See Also

vid_scroll_up()
vid_scroll_down()
vid_scroll_right()
vid_scroll_up_no_clear()
vid_scroll_down_no_clear()
vid_scroll_left_no_clear()
vid_scroll_right_no_clear()

DeskMate Technical Reference

Video Manager

vid_scroll_left_no_clear (Xtext)

int Xtext;

vid_scroll_left_no_clear scrolls the screen within the current clipping region left the specified number of world coordinate units. The right part of the clipping region is not cleared.

Input Parameters

Xtext is the number of world coordinate units to scroll the screen left.

Return Value

None.

Special Notes

If **Xtext** is less than or equal to zero, no action is performed.

See Also

vid_scroll_up()
vid_scroll_down()
vid_scroll_left()
vid_scroll_right()
vid_scroll_up_no_clear()
vid_scroll_down_no_clear()
vid_scroll_right_no_clear()

DeskMate Technical Reference

Video Manager

vid_scroll_right (Xext)

int Xext;

vid_scroll_right scrolls the screen within the current clipping region right the specified number of world coordinate units. The left part of the clipping region is cleared to the current background color.

Input Parameters

Xext is the number of world coordinate units to scroll the screen right.

Return Value

None.

Special Notes

If **Xext** is less than or equal to zero, no action is performed.

Example

```
/* scroll right 5 characters */  
vid_scroll_right( 5 * CHAR_XEXT );
```

See Also

vid_scroll_up()
vid_scroll_down()
vid_scroll_left()
vid_scroll_up_no_clear()
vid_scroll_down_no_clear()
vid_scroll_left_no_clear()
vid_scroll_right_no_clear()

DeskMate Technical Reference

Video Manager

vid_scroll_right_no_clear (Xtext)
int Xtext;

vid_scroll_right_no_clear scrolls the screen within the current clipping region right the specified number of world coordinate units. The left part of the clipping region is not cleared.

Input Parameters

Xtext is the number of world coordinate units to scroll the screen right.

Return Value

None.

Special Notes

If **Xtext** is less than or equal to zero, no action is performed.

See Also

vid_scroll_up()
vid_scroll_down()
vid_scroll_left()
vid_scroll_right()
vid_scroll_up_no_clear()
vid_scroll_down_no_clear()
vid_scroll_left_no_clear()

DeskMate Technical Reference

Video Manager

vid_scroll_up (Yext)
int Yext;

vid_scroll_up scrolls the screen within the current clipping region up the specified number of world coordinate units. The bottom part of the clipping region is cleared to the current background color.

Input Parameters

Yext is the number of world coordinate units to scroll the screen up.

Return Value

None.

Special Notes

If **Yext** is less than or equal to zero, no action is performed.

Example

```
/* scroll up one line */  
vid_scroll_up( CHAR_YEXT );
```

See Also

vid_scroll_down()
vid_scroll_right()
vid_scroll_left()
vid_scroll_up_no_clear()
vid_scroll_down_no_clear()
vid_scroll_left_no_clear()
vid_scroll_right_no_clear()

DeskMate Technical Reference

Video Manager

vid_scroll_up_no_clear (Yext)

int Yext;

vid_scroll_up_no_clear scrolls the screen within the current clipping region up the specified number of world coordinate units. The bottom of the clipping region is not cleared.

Input Parameters

Yext is the number of world coordinate units to scroll the screen up.

Return Value

None.

Special Notes

If **Yext** is less than or equal to zero, no action is performed.

See Also

vid_scroll_up()
vid_scroll_down()
vid_scroll_left()
vid_scroll_right()
vid_scroll_down_no_clear()
vid_scroll_left_no_clear()
vid_scroll_right_no_clear()

DeskMate Technical Reference

Video Manager

vid_set_attrs (pVID_ATTRS)
VID_ATTRS *pAttrs;

vid_set_attrs copies a VID_ATTRS structure into the CSR. This will set all the CSR video attributes.

Input Parameters

pVID_ATTRS is a pointer to a VID_ATTRS structure in your data segment.

Return Value

None.

Example

```
VID_ATTRS  Attrs =
{
    ENABLED,
    0, 0,
    VID_BLOCK_CURSOR,
    NULL,          /* cursor length */
    NULL,          /* underline cursor color */
    NULL,          /* seg of defined cursor */
    NULL,          /* offset of defined color */
    NULL,          /* xfocal */
    NULL,          /* yfocal */
    NORMAL,        /* char attr */
    COLOR1,        /* bg color */
    COLOR2,        /* fg color */
    LINE_SOLID,    /* line type */
    LINE_WIDTH1,   /* line width */
    COLOR4,        /* line fg color */
    PATTERN2,      /* pattern used for fills */
    CHAR_XEXT,     /* char x extent */
    CHAR_YEXT,     /* char y extent */
    COLOR1,        /* line fg for brushes */
};

/* set core video attributes to their initialized state */
vid_set_attrs( Attrs );
```

See Also

vid_move_cursor()
vid_set_cursor_type()
vid_set_char_attr()
vid_get_attrs()
vid_set_line_attr()
vid_set_colors()
vid_set_pattern()
vid_set_char_size()

DeskMate Technical Reference

Video Manager

vid_set_char_attr (CharAttr)

char CharAttr;

vid_set_char_attr sets the current character attribute for subsequent character output calls.

Input Parameters

CharAttr is one (or more) of the defined character attributes that the video routines support.

Return Value

None.

Special Notes

NORMAL, BOLD, UNDERLINE, INVERSE, and GRAYED and TRANSPARENT are currently supported character attributes. ITALIC is not supported.

Multiple attributes are created by "or-ing" attributes together.

Example

```
vid_set_char_attr( BOLD );  
vid_put_string( "This is BOLD" );  
vid_set_char_attr( UNDERLINE | BOLD );  
vid_put_string( "This is UNDERLINE BOLD" );
```

DeskMate Technical Reference

Video Manager

vid_set_char_size (Xext, Yext)

int Xext;

int Yext;

vid_set_char_size defines the world coordinate size of the system font.

Input Parameters

Xext is an integer x extent of the system font characters.

Yext is an integer y extent of the system font characters.

Return Value

None.

Special Notes

The CSR defaults the character size to 100 CHAR_XEXT by 220 CHAR_YEXT. This size allows the display to be exactly 80 by 25 characters with the default world and viewport settings. It is also the optimal size for speed under the default world and viewport. If an application changes the world, viewport, and/or character size such that the display is no longer exactly 80 by 25 characters, then characters will be displayed considerably slower.

DeskMate Technical Reference

Video Manager

vid_set_clip (pMAPRECT)
MAPRECT *pMAPRECT;

vid_set_clip sets the clip region. The origin and extents are in world coordinates.

Input Parameters

pMAPRECT is a pointer to a MAPRECT structure in your data segment.

Return Value

None.

Special Notes

After activating a window the origin of the clipping rectangle will always be (0,0). The clip rectangle will be clipped to the device display surface. For example, if you tried to set a clip region larger than the screen, the values will be clipped to stay on the screen. If **vid_get_clip** is performed you will get the clipped values.

Example

```
MAPRECT    WorldMR;  
  
/* set clip region equal to world region */  
vid_get_world( &WorldMR );  
vid_set_clip( &WorldMR );
```

See Also

vid_get_clip()
vid_get_world()

DeskMate Technical Reference

Video Manager

vid_set_clip_to_window ()

vid_set_clip_to_window sets the clipping region to be equal to the boundaries of the current window. This is the default condition upon activating a window. Thus **vid_set_clip_to_window** is only useful if you have performed **vid_set_clip**.

Input Parameters

None.

Return Value

None.

Special Notes

vid_set_clip_to_window will always set the origin of the clip region to (0,0).

See Also

vid_set_clip()

DeskMate Technical Reference

Video Manager

vid_set_colors (BackgroundColor, ForegroundColor)

int BackgroundColor;

int ForegroundColor;

vid_set_colors sets the current background and foreground color for subsequent character output to the screen.

Input Parameters

BackgroundColor is one of the defined colors for the desired background color.

ForegroundColor is one of the defined colors for the desired foreground color.

Return Value

None.

Special Notes

The defines for the colors are: COLOR1 - COLOR16. **vid_set_colors** does not affect line colors.

Example

```
vid_set_colors( COLOR1, COLOR2 );
```

DeskMate Technical Reference

Video Manager

vid_set_cursor_type (Type, Extent, Color, DeviceX, DeviceY, pCursorDef)

char Type;

int Extent;

char Color;

int DeviceX, DeviceY, *pCursorDef;

vid_set_cursor_type sets the current cursor type.

Input Parameters

Type is the cursor type define code.

Extent is only used for VID_LINE_CURSOR type, **Extent** can equal 1-8000.

Color is only used for VID_LINE_CURSOR type. **Color** can be COLOR1 - COLOR16.

DeviceX and **DeviceY** describe the device pixel relative to the text cursor. This is only used for VID_DEFINED_CURSOR.

pCursorDef is a pointer to a 32 integer array containing 16 AND masks and 16 XOR masks with a range 0-FFFF; this is only used for VID_DEFINED_CURSOR.

Return Value

None.

Special Notes

VID_BLOCK_CURSOR, VID_BAR_CURSOR, VID_LINE_CURSOR, and VID_DEFINED_CURSOR are available define codes for the cursor.

pCursorDef is a map of a 16 x 16 pixel cursor. Each pixel is represented by an AND bit mask and a XOR bit mask. Different combinations produce the following results:

AND bit mask	XOR bit mask	Result
0	0	COLOR3
0	1	COLOR4
1	0	pixel unchanged
1	1	pixel inverted

Example

```
/* This is a sample showing how to define a new mouse */
/* pointer or a defined text cursor */
/* The sample uses the masks for the standard mouse pointer */
/* Note that this definition id device dependent, that is, */
/* it will appear to shrink on high resolution devices. */
int cursorDef[] =
{
    0x3FFF,    /* line 1 start of and_mask */
    0x1FFF,    /* line 2 */
    0x0FFF,    /* line 3 */
    0x07FF,    /* line 4 */
    0x03FF,    /* line 5 */
    0x01FF,    /* line 6 */
    0x00FF,    /* line 7 */
    0x007F,    /* line 8 */
    0x003F,    /* line 9 */
}
```

DeskMate Technical Reference

Video Manager

```
0x001F,    /* line 10 */
0x00FF,    /* line 11 */
0x00FF,    /* line 12 */
0x007F,    /* line 13 */
0x183F,    /* line 14 */
0xFC1F,    /* line 15 */
0xFE1F,    /* line 16 end of and_mask */
0x0000,    /* line 1 start of xor_mask */
0x4000,    /* line 2 */
0x6000,    /* line 3 */
0x7000,    /* line 4 */
0x7800,    /* line 5 */
0x7C00,    /* line 6 */
0x7E00,    /* line 7 */
0x7F00,    /* line 8 */
0x7F80,    /* line 9 */
0x7FC0,    /* line 10 */
0x7FE0,    /* line 11 */
0xFF00,    /* line 12 */
0x6700,    /* line 13 */
0x3800,    /* line 14 */
0x1C00,    /* line 15 */
0x0000,    /* line 16 end of xor_mask */
};

vid_set_cursor_type( VID_BAR_CURSOR );
.
.
.
vid_set_cursor_type( VID_LINE_CURSOR, CHAR_XEXT, COLOR3 );
.
.
.
/* put the 'hot spot' at the upper left corner of cursorDef */
vid_set_cursor_type( VID_DEFINED_CURSOR, NULL, NULL, 0, 0, cursorDef );

or

/* put the 'hot spot' in the middle of cursorDef */
vid_set_cursor_type( VID_DEFINED_CURSOR, NULL, NULL, 8, 8, cursorDef );

or

/* put the 'hot spot' in the bottom left corner of cursorDef */
vid_set_cursor_type( VID_DEFINED_CURSOR, NULL, NULL, 0, 16, cursorDef );
```

DeskMate Technical Reference

Video Manager

vid_set_line_attr (Type, Width, ForegroundColor, BackgroundColor)

char type;
char width;
char ForegroundColor;
char BackgroundColor;

vid_set_line_attr sets the line attributes to be used in all line draws and the edges of all objects.

Input Parameters

Types are: LINE_SOLID, LINE_INVISIBLE, LINE_DOTTED, LINE_DASHED, LINE_DOT_DASHED.

Widths are: LINE_WIDTH1 - LINE_WIDTH5, or LINE_BRUSH1 - LINE_BRUSH20.

ForegroundColor can be COLOR1 - COLOR16, or COLOR_XOR.

BackgroundColor can be COLOR1 - COLOR16, only used for brushes.

Return Value

None.

Special Notes

LINE_WIDTH1 is the thin line which is always one pixel wide. This line is good for tight borders such as frames.

LINE_WIDTH2 is the best and most even looking line since it is optimized for each video device (on a 1000 it is 2x1 pixels). The width of this line is device dependent. All line types are supported under this width.

LINE_WIDTH3, LINE_WIDTH4, and LINE_WIDTH5 are varying sized filled lines. LINE_SOLID is the only line type supported for these line widths. LINE_WIDTH3 is approximately 50 world coordinates wide. LINE_WIDTH4 is approximately 75 world coordinates wide. LINE_WIDTH5 is approximately 100 world coordinates wide.

The COLOR_XOR attribute is considered to be a color attribute. When COLOR_XOR is used then the line width is ignored and LINE_WIDTH1 is used.

vid_set_line_attr does not affect the character foreground and background colors.

A line has either a LINE_WIDTH or a LINE_BRUSH define but cannot have both. Lines only use **ForegroundColor** brushes use **ForegroundColor**, **BackgroundColor** and the current fill pattern.

Example

```
vid_set_line_attr( LINE_SOLID, LINE_WIDTH1, COLOR1 );  
:  
:  
vid_set_line_attr( LINE_DASHED, LINE_WIDTH3, COLOR2 );
```


DeskMate Technical Reference Video Manager

vid_set_palette (pVID_PALETTE)
VID_PALETTE *pVID_PALETTE;

vid_set_palette sets the palette colors.

Input Parameters

pVID_PALETTE is a pointer to a VID_PALETTE structure.

Return Value

None.

Special Notes

The following table gives RGB values for the basic 16 colors. Some devices such as EGA and VGA support more than the basic 16 colors. The valid range for the Red, Green and Blue values is not limited to the values in the table but can be any value from 0 to 255.

	Color Values		
	Red	Green	Blue
black	0	0	0
blue	0	0	128
green	0	128	0
cyan (pale blue)	0	128	128
red	128	0	0
magenta (purple)	128	0	128
yellow (appears as orange)	128	128	0
gray	128	128	128
dark gray	64	64	64
bright blue	0	0	192
bright green	0	192	0
bright cyan (bright pale blue)	0	192	192
bright red	192	0	0
bright magenta (bright purple)	192	0	192
bright yellow (appears as normal yellow)	192	192	0
white	192	192	192

Bug

There is a bug such that you should never set palettes 16 to 255 or the busy icon may malfunction. Only palettes 0 to 15 should be set.

Example

```
VID_PALETTE palette =  
{  
    COLOR1,    /* set the first palette */  
    128,       /* Red setting is 128 */  
    0,         /* Green setting is 0 */
```

DeskMate Technical Reference Video Manager

```
128          /* Blue setting is 128 */  
};          /* the above RGB values will produce purple */
```

Note: the RGB format is provided for device independence. RGB values can be any value from 0 - 255 but if the device does not support the particular hue specified then the "best fit" hue is used. For example assume that Red and Green are left at 0 and only the blue value is changed.

```
CGA /* black plus 2 shades of blue */  
0 - 127 = black  
128 - 191 = blue  
192 - 255 = bright blue
```

```
EGA /* black plus 3 shades of blue */  
0 - 63 = black  
64 - 127 = dark blue  
128 - 191 = blue  
192 - 255 = bright blue
```

```
VGA /* black plus 63 shades of blue */  
0 - 3 = black  
4 - 7 = darkest shade of blue  
8 - 11 = 2nd darkest shade of blue  
.  
.  
128 - 131 = blue  
.  
.  
248 - 251 = 2nd brightest shade of blue  
252 - 255 = brightest blue
```

Thus for a blue value of 77, CGA would give black, EGA would give dark blue and VGA would yield a shade of blue which would be 76 units of blue on a scale of 0 to 255.

DeskMate Technical Reference Video Manager

vid_set_pattern (Pattern)
char Pattern;

vid_set_pattern sets the pattern to be used in all fills and line brushes.

Input Parameters

Pattern is a defined type with range PATTERN1 - PATTERN20.

Return Value

None.

Example

```
vid_set_pattern( PATTERN12 );      /* diamonds pattern */  
vid_set_pattern( PATTERN16 );      /* shingles pattern */
```

Bug

The pattern attribute is destroyed when an ellipse, polyline, polygon, rectangle, or arc are rendered.

See Also

fvid_draw_form()
vid_draw_form()

DeskMate Technical Reference

Video Manager

vid_set_viewport (pMAPRECT)
MAPRECT *pMAPRECT;

vid_set_viewport sets the viewport by the device coordinates specified in **pMAPRECT**.

Input Parameters

pMAPRECT is a pointer to a **MAPRECT** structure in your data segment.

Return Value

None.

Special Notes

win_activate makes calls to **vid_set_world** and **vid_set_viewport** to set up your window. Normally the application programmer does not need to call **vid_set_world**, **vid_get_world**, **vid_set_viewport** or **vid_get_viewport**. The extents must not equal zero.

Example

```
MAPRECT      DeviceMR;  
DeviceMR.xorg = 0;  
DeviceMR.yorg = 0;  
DeviceMR.xext = 640;  
DeviceMR.yext = 200;  
  
/* set viewport to be 640 device coordinates */  
/* by 200 device coordinates */  
  
vid_set_viewport( &DeviceMR );
```

See Also

win_activate()
vid_set_world()
vid_get_world()
vid_get_viewport()

DeskMate Technical Reference Video Manager

vid_set_world (pMAPRECT)
MAPRECT *pMAPRECT;

vid_set_world sets the world boundaries using the world coordinates in **pMAPRECT**.

Input Parameters

pMAPRECT is a pointer to a MAPRECT structure in your data segment.

Return Value

None.

Special Notes

win_activate makes calls to **vid_set_world** and **vid_set_viewport** to set up your window. Normally the application programmer does not need to call **vid_set_world**, **vid_get_world**, **vid_set_viewport** or **vid_get_viewport**. The extents must not equal zero or divide overflow will occur.

Example

```
MAPRECT      WorldMR;  
WorldMR.xorg = 0;  
WorldMR.yorg = 0;  
WorldMR.xext = 8000;  
WorldMR.yext = 5500;  
  
/* set world to be 8000 world coordinates wide */  
/* and 5500 world coordinates in height */  
vid_set_world( &WorldMR );
```

See Also

win_activate()
vid_get_world()
vid_set_viewport()
vid_get_viewport()

DeskMate Technical Reference

Video Manager

```
int vid_wcx_to_dcx ( WorldCordX )  
int WorldCordX;
```

vid_wcx_to_dcx converts a world coordinate x (relative to the active window) to a device coordinate x (relative to the screen).

Input Parameters

WorldCordX is the world coordinate x to convert.

Return Value

<int>
A device coordinate x.

Example

```
int   WorldX;  
int   DeviceX;  
  
/* convert world coordinate in WorldX to */  
/* device coordinate stored in DeviceX */  
  
DeviceX = vid_wcx_to_dcx( WorldX );
```

See Also

vid_wcy_to_dcy()
vid_dcx_to_wcx()
vid_dcy_to_wcy()

See the Development Guide Getting Started section for a discussion of the world coordinate system.

DeskMate Technical Reference

Video Manager

```
int vid_wcx_to_nwcx ( WorldCordX )  
int WorldCordX;
```

vid_wcx_to_nwcx converts a world coordinate x to a normalized world coordinate x.

Input Parameters

WorldCordX is the world coordinate x to convert.

Return Value

A normalized world coordinate x.

Example

```
int    NormalizedX;  
int    WorldX;  
  
NormalizedX = vid_wcx_to_nwcx( WorldX );
```

See Also

vid_wcy_to_nwcy()

See the Development Guide Getting Started section for a discussion of the world coordinate system.

DeskMate Technical Reference

Video Manager

```
int vid_wcy_to_dcy ( WorldCordY )  
int WorldCordY;
```

vid_wcy_to_dcy converts a world coordinate y (relative to the active window) to a device coordinate y (relative to the screen).

Input Parameters

WorldCordY is the world coordinate y to convert.

Return Value

<int>
A device coordinate y.

Example

```
int   WorldY;  
int   DeviceY;  
  
/* convert world coordinate in WorldY to */  
/* device coordinate stored in DeviceY */  
DeviceY = vid_wcy_to_dcy( WorldY );
```

See Also

vid_wcx_to_dcx()
vid_dcx_to_wcx()
vid_dcy_to_wcy()

See the Development Guide Getting Started section for a discussion of the world coordinate system.

DeskMate Technical Reference

Video Manager

```
int vid_wcy_to_nwcy ( WorldCordY )  
int WorldCordY;
```

vid_wcy_to_nwcy converts a world coordinate y to a normalized world coordinate y.

Input Parameters

WorldCordY is the world coordinate y to convert.

Return Value

A normalized world coordinate y.

Example

```
int    NormalizedY;  
int    WorldY;  
  
NormalizedY = vid_wcy_to_nwcy( WorldY );
```

See Also

vid_wcx_to_nwcx()

See the Development Guide Getting Started section for a discussion of the world coordinate system.

Window Manager

"Window Manager"

Table of Contents

General Description/Notes	26- 1
Structure	26- 2
win_activate	Makes a specified window active.....26- 3
win_close	Removes a window from the system.....26- 4
win_get_active	Gets the currently active window.....26- 5
win_group_end	Deletes base window and current group.....26- 6
win_group_init	Initiates a new window group.....26- 7
win_open	Creates and initializes a new window.....26- 8

DeskMate Technical Reference Window Manager

General Description/Notes

The window manager manages the creation, deletion, and activation of consumer defined windows. A window is basically a video context or a screen state. When a window is created, it is given a default context. The context includes clip region, cursor position and state. When the newly created window is activated, the screen state is placed into the context of that window. At this point, the window context may be changed and the screen may be written to using that context. When another window is activated, the current window context is saved then the screen state will be changed according to the new window context.

If a clip region is defined within a window, window events are confined to the clip region.

There is a fixed 2K buffer used to track windows and hot spots. Each newly created window uses 40 hex bytes, each hot spot uses 12 hex bytes. For example:

MAX window buffer size	=	2048 bytes (39 windows)
each win_group_init	=	- 68 bytes (done by each csr_init)

		1980 bytes
menu bar entry	=	- 16 bytes (hot spot entry)

		1964 bytes
each menu button	=	x bytes (No. buttons * 32)

		1964 bytes
each menu item w/accel	=	x bytes (No. items * 16)

		1964 bytes

		1964 bytes

For each menu button F9 or F10 it takes one hot spot. (16 bytes each)
Each window created by an application (win_open) is one window. (52 bytes)
Each component created/opened (cmp_open (non push button)) is one hot spot. (16 bytes)
Each push button with DEFAULT_ACCELERATOR is 2 hot spots. (32 bytes)
Each push button with other than DEFAULT_ACCELERATOR is one hot spot. (16 bytes)

$$\text{no. windows available} = \frac{\text{BUFFER} - ((\text{num windows} * 52) + (\text{num hot spots} * 16))}{52}$$
$$\text{no. hot spots available} = \frac{\text{BUFFER} - ((\text{num windows} * 52) + (\text{num hotspots} * 16))}{16}$$

Note: You must allow for one window entry to exist (free), to use the component manager.

DeskMate Technical Reference Window Manager

Typically, the CSR will prevent all windows from receiving mouse hold events except for the window in which the button down event was detected. This is accomplished within the CSR by recording the handle of the current window when a button down is detected. All subsequent mouse hold events are returned only if the currently active window has that handle

This allows two different windows to receive all mouse hold events. This occurs when one window receives a button down event, is then closed, and then another window is created. This occurs because the handle is freed when the first window is closed, making it available for the next created window. This problem will only occur if a window is created, reads a button down event, is then closed prior to the termination of that hold sequence, and a new window is then created that processes events.

DeskMate Technical Reference Window Manager

Structure

```
/*-----*/
/*                                          */
/* CSRBASE.H contains the Window structure */
/*                                          */
/*-----*/

struct maprect_defn
{
    int    xorg; /* rectangle x origin */
    int    yorg; /* rectangle y origin */
    int    xext; /* rectangle x extent */
    int    yext; /* rectangle y extent */
};
typedef struct maprect_defn MAPRECT;
```

DeskMate Technical Reference

This page intentionally left blank.

DeskMate Technical Reference

Window Manager

win_activate (Handle)

int Handle;

win_activate deactivates the currently active window and activates the specified window.

Input Parameters

Handle is the handle assigned to the desired window when the window was created.

Return Value

<int>

CSR_ERROR if the handle refers to a non-existent window or a window that has been closed.

Special Notes

The video state of the original window is preserved and the state of the new window is restored.

Example

```
int ReturnCode;
int WindowHandle;

/* Make the window with the handle WindowHandle */
/* the currently active window */
ReturnCode = win_activate( WindowHandle );
```


DeskMate Technical Reference Window Manager

win_close (Handle)
int Handle;

win_close removes a window from the system.

Input Parameters

Handle is the handle assigned to the desired window when the window was created.

Return Value

None.

Special Notes

If the window specified is the currently active window, then after the window is removed, the base window becomes the active window. Once a window is closed, it cannot be reactivated.

DeskMate Technical Reference Window Manager

win_get_active ()

win_get_active returns the handle of the currently active window.

Input Parameters

None.

Return Value

<int>

The handle of the currently active window.

Example

```
int. ReturnCode;  
/* get current active window handle */  
ReturnCode = win_get_active();
```

DeskMate Technical Reference

Window Manager

win_group_end ()

win_group_end deletes the base window and all other windows in the current window group.

Input Parameters

None.

Return Value

None.

Special Notes

The previously active window group (if any) is reactivated after the current window group is deleted.

Example

```
/* delete the current base window */  
win_group_end();
```

DeskMate Technical Reference Window Manager

win_group_init ()

win_group_init preserves the current active window and initiates a new window group.

Input Parameters

None.

Return Value

The handle to the new base window of the new window group.

Special Notes

The new group will initialize the first window level above the currently highest window level to the base screen state. This base window level can only be deleted with a call to **win_group_end**. There is a fixed 2K buffer used to track windows and hot spots. Each **win_group_init** takes 68 bytes from the buffer. **win_group_end** will replace the used 68 bytes, and all the other windows and hotspots that are created within the group session.

Example

```
int NewBaseWindow;  
  
/* make a new base window group active */  
NewBaseWindow = win_group_init();
```

DeskMate Technical Reference Window Manager

win_open (pMAPRECT)
WINDOW *pMAPRECT; } *Incorrect. There is no such WINDOW structure.
Should be MAPRECT *pMAPRECT.*

win_open creates and initializes a new window work area. *See Dmdecl.h.*

Input Parameters

pMAPRECT is a pointer to a MAPRECT structure.

Return Value

<int>

The handle assigned to the new window.

Special Notes

No part of the window becomes visible after making **win_open**, and the current window remains the active window (use **win_activate** to make the new window the active window).
The new window's cursor defaults to position (0,0) in the window and is turned off, it's current character attribute defaults to NORMAL.

See Also

win_activate()

Font Resource

Table of Contents

General Description/Notes	27- 1
Structures/Defines	27- 2
eform_bind_end	Terminates the forms resource.....27- 5
eform_bind_init	Links to the forms resource..... 27- 6
font_bind_end	Terminates the font resource.....27- 7
font_bind_init	Links to the font resource.....27- 8
font_disk_swap	Informs font resource of disk swap.....27- 9
font_face_lock	Disable auto de-allocation of font face....27-10
font_face_support ..	Provides support for font face types.....27-11
font_face_unlock ...	Enables auto de-allocation of font face....27-12
font_get_bitmap	Gets bitmap image for font character.....27-13
font_get_cell	Gets font character dimensions.....27-14
font_get_dpi	Gets resolution of output device.....27-15
font_get_face	Gets a font from a list of fonts.....27-16
font_get_style	Gets style info attributes.....27-17
font_get_width	Gets the width of a character string.....27-18
font_plot_bitmap ...	Plots bit image of font char to buffer....27-19
font_set_buffer	Allocates buffers for font caching.....27-20
font_set_dpi	Set resolution of output device.....27-21

DeskMate Technical Reference Font Resource

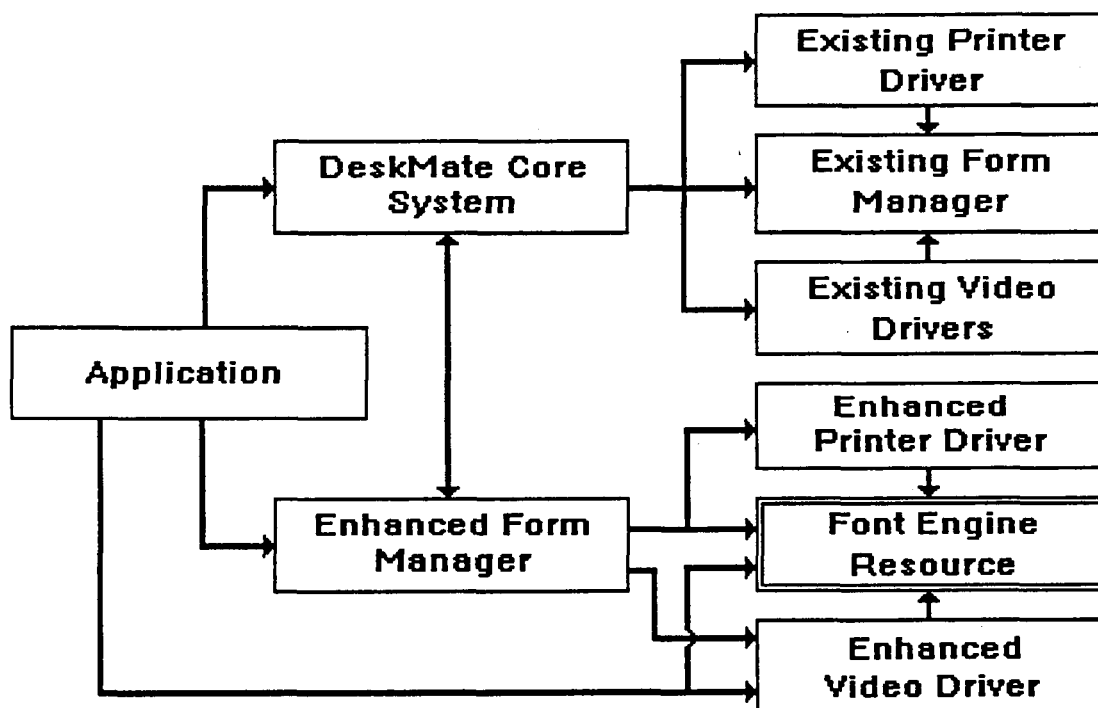
General Description/Notes

The font engine resource provides font rasterization, management of font files and caching.

There are two mechanisms by which you can have fonts rendered.

1. To render all fonts through the enhanced form manager in which you can render to screen as well as the printer.
2. To render through the font engine resource and enhanced video driver, this provides faster rendition of fonts. This method does not provide rendition to the printer.

The following diagram reflects the system integration of the font engine and associated support resources in the DeskMate environment.



The following functions are intended for use by the DeskMate environment and application for scalable font characters.

The following sections describe the parameter requirements for the functions. These functions are primarily intended to be used by the DeskMate environment, however, an application is allowed to call any or all of them directly.

DeskMate Technical Reference

Font Resource

Structures/Defines

```

/*-----*/
/* Structure Components and definitions required to interface to the */
/* FONT resource. */
/*-----*/

/* possible returns after a function call to the font resource */
DMF_OK          0          /* success in completing the call */
DMF_ERROR       -1          /* unsuccessful in completing the call */
DMF_NOFILES     -2          /*--flag indicating no more file found */

DMF_FREE        0          /*--flag used to free font_set_buffer */

/* font types requiried flags. */
FF_BITMAP       0x00       /* internal use only */
FF_RASTERIZE    0x01
FF_RESIDENT     0x02
FF_SYSTEM       0x04

/* The following are the basic font styles that are */
/* supported under the DeskMate environment. Font style */
/* bit flags for requesting styles. */
DFS_NORMAL      0x01       /* use normal font styles. */
DFS_UNDERLINE   0x02       /* underline the character */
DFS_BOLD        0x04       /* use bold font style. */
DFS_ITALIC      0x08       /* use italic font style. */
DFS_SUPERSCRIPT 0x10       /* superscript the character */
DFS_SUBSCRIPT   0x20       /* subscript the character */
DFS_GRAYED      0x40       /* grayed character are to be outputted */
DFS_HOLLOW      0x80       /* an outline of the character. */

DFS_ALL_STYLES  0x00ff     /* all the font styles are available */

DFS_MATRIX      0x100      /* Matrix data provided is to be used. */

/* This structure is used to control what font faces are */
/* supported and to indicate the particular font face information */
/* required by the font engine. */
struct face_defn {
    unsigned char  bType;      /* font type: resident,rasterize */
    int            request;    /* */
    unsigned char  nResident;  /* number of resident fonts found */
    unsigned char  nRasterize; /* number of rasterize fonts found */
};
typedef struct face_defn FACE ;
Note.
Request element data contains the integer number of the particular face required if known for the
function call font_get_face. This element is also incremented every time font_get_face is
called to index the next face.

/* This matrix structure provides the four values for controlling */
/* the position of how the character is to be rasterized. */
struct matrix_defn {
    long  a11;      /* matrix value for row 1 column 1 */
    long  a12;      /* matrix value for row 1 column 2 */
    long  a21;      /* matrix value for row 2 column 1 */
    long  a22;      /* matrix value for row 2 column 2 */
};
typedef struct matrix_defn MATRIX ;

struct font_face_defn {

```

DeskMate Technical Reference

Font Resource

```

unsigned char  bType;          /* font type: resident, rasterize */
unsigned char  state;          /* font locking state, internal use only*/
long           bFontClass;     /* font class to which it belongs */
char           face_name[28];  /* null term font face string name*/
unsigned int   bStyleAttrs;    /* style of character bitmap to */
                                /* generate */
unsigned int   PtSize;         /* point size required, 1-1024 */
unsigned int   pitch;          /* char. pitch, 150 would expand by 1.5 */
unsigned int   rotation;       /* degree of rotate, clockwise from vert*/
unsigned int   spacing;        /* in DeskMate world units for mono. */
                                /* 0=prop */
unsigned int   cpi;            /* cpi in 100 units, resident fonts only*/
unsigned int   bStyle;         /* styles capable of this font face */
MATRIX         matrix;        /* transformation matrix for generating */
                                /* character */
};
typedef struct font_face_defn FONT_FACE ;
FONT_FACE.PtSize - The maximum point size supported is 1024.

struct font_cell_defn {
    unsigned int height;        /* height of font character cell, */
    unsigned int leading;       /* leading size of font, */
    int          ascii_ch;       /* char of bitmap cell required */
    int          width;          /* width of font character */
    int          wc_width;       /* width of bitmap character */
    int          widthOffset;    /* offset of width */
    int          wc_height;      /* total height of bitmap height */
    int          ascent;        /* height above the baseline */
};
typedef struct font_cell_defn FONT_CELL ;
Note
All values are returned in DeskMate world co-ordinate units.

struct output_style_info_defn {
    char          ulThickness;    /* underline thickness in pixels */
    char          ulOffset;       /* underline offset in pixels */
    char          superOffset;    /* superscript offset as % of point size*/
    char          subOffset;      /* subscript offset as % of point size */
};
typedef struct output_style_info_defn OUTPUT_STYLE_INFO ;

struct bitmap_defn {
    int          ascii_ch;        /* character of bit map */
    unsigned int pixel_width;     /* width of bit map in pixels */
    unsigned int pixel_height;    /* height of bit map in pixels */
    unsigned int cell_wcxext;     /* width of bit map in DeskMate */
                                /* world coordinates */
    unsigned int xOffset;         /* xOffset of character origin. */
    unsigned int yOffset;        /* yOffset of character origin. */
    unsigned char far *fpBitmap;  /* far pointer to start of bitmap*/
};
typedef struct bitmap_defn BITMAP;

```

BITMAP.fpBitmap - points to a buffer in which to store the bitmap. The length of this buffer should NOT be less than ((pixel_width/8) * pixel_height) or the buffer will be overrun. The pixel_width and pixel_height can be smaller than the actual character, therefore asking for a particular section of a character at a time. The bitmap is in the format of 640 * 350 * 2 color, where 0 = white and 1 = black.

Note:

The values are returned in DeskMate world co-ordinate units.

DeskMate Technical Reference Font Resource

```

/*----- */
/* ENHANCED FORM FONT interface structures and definitions. */
/* DMFNTFRM.H */
/*----- */

/* Fonts are of type FORM_OTHER */
/* FORM_OTHER is put into ELEMENT.type */
FORM_U_FONT_TYPE 'F' /* value to be put into ELEMENT.mod */
FORM_U_FONT_TYPE_INTERNAL 'f' /* internal font format */

/* The FONT structure is used by the application */
/* to interface with the enhanced video driver. */
struct font_defn
{
    FONT_FACE far *fpFaceInfo; /* far ptr to font face info */
    BBOX_INFO bbox_info; /* internal use only */
    char fgnd_color; /* character foreground color */
    char reserved;
    int string_xorg; /* world cord xorg, */
    /* window relative */
    int string_yorg; /* world cord yorg, */
    /* window relative */
    int nChars; /* number of chars in string */
    char far *fpString; /* far ptr to the text string */
    char far *fpEscapement; /* far ptr to escapement list */
};
typedef struct font_defn FONT;

/* The FORM_FONT structure is used by the application */
/* to add font elements to the enhanced form manager. */
struct form_font_defn
{
    FORM_U u_header;
    FONT info;
};
typedef struct form_font_defn FORM_FONT;

struct form_font_attr_defn
{
    char text_color;
    char text_attr;
    char text_rot;
    char font;
    char line_fgnd_color;
    char line_style;
    char line_width;
    char fill_pattern;
    char bgnd_color;
    char fgnd_color;
    char bev_pad;
    char line_bgnd_color;
    char pad;
    FONT_FACE far *fpFaceInfo; /* far pointer to font face info */
};
typedef struct form_font_attr_defn FORM_FONT_ATTR;

```

DeskMate Technical Reference

Font Resource

eform_bind_end ()

eform_bind_end terminates the forms resource loaded by **eform_bind_init** and unbinds any resources loaded by the enhanced (far) forms resource.

Input Parameters

None.

Return Value

None.

Special Notes

eform_bind_end requires the application to link with DeskMate 03.05 or later libraries (DM.LIB or DMMED.LIB).

DeskMate Technical Reference

Font Resource

eform_bind_init (Font)

eform_bind_init loads and links to the forms resource as an enhanced (far) forms resource. This functionality is only available in for use in DeskMate 03.05 and later.

Input Parameters

<char>

If **Font** is **FALSE** the enhanced form manager will not support fonts. The font resource and enhanced video driver will not be loaded into memory.

If **Font** is **TRUE** the enhanced form manager will support fonts, and the enhanced video driver will be loaded into memory.

Return Value

<int>

DM_OK if everything was loaded correctly.

DM_ERROR if the forms resource could not be loaded.

Special Notes

eform_bind_init requires the application to link with DeskMate 03.05 or later libraries (DM.LIB or DMMED.LIB).

DeskMate Technical Reference

Font Resource

void font_bind_end (void)

font_bind_end terminates the font resource loaded by **font_bind_init** and unbinds any resources loaded by the font resource.

Input Parameters

None.

Return Value

None.

DeskMate Technical Reference

Font Resource

int **font_bind_init** (void)

font_bind_init loads and links to the font resource.

Input Parameters

None.

Return Value

DMF_ERROR if an error occurred in loading the resource.

DeskMate Technical Reference

Font Resource

void font_disk_swap (void)

font_disk_swap notifies the font engine resource of a potential disk swapping. At this point it will be necessary to close all open files and precess tasks as necessary.

Input Parameters

None.

Return Value

None.

DeskMate Technical Reference

Font Resource

```
int font_face_lock ( fpFONT_FACE )  
FONT_FACEfar      *fpFONT_FACE;
```

font_face_lock allows the caller to lock a particular font face, stopping the font engine from automatically swapping out the font faces resource (i.e. cached character bitmaps, width table) and other associated resources. The default state of a font face is unlocked.

Input Parameters

fpFONT_FACE is passed containing the relevant font face description.

Return Value

DMF_OK (0) indicates success in locking the font face.

DMF_ERROR (-1) indicates unsuccessful in locking the font face.

DeskMate Technical Reference

Font Resource

```
int font_face_support ( fpFACE )  
FACE far *fpFACE;
```

font_face_support provides the font engine with the appropriate information on what font faces to support. **font_face_support** will read all the necessary files from disk to setup its internal font data structures for it to rasterize a font character.

Input Parameters

fpFACE is a far pointer to a data structure of type **FACE**.

Return Value

An integer is returned indicating the number of font faces found. The data structure **FACE** (**fpFACE**) will be updated with the necessary information, number of resident and rasterize font faces found.

Special Notes

Upon entry within the **FACE** structure, the element **bType** must be set to indicate the font type support required.

FF_RESIDENT	0x01
FF_RASTERIZE	0x02
FF_SYSTEM	0x04

Request element data contains the read/update defines when a call is made to **font_face_support** but an integer index is kept for function call **font_get_face**.

There will always be a default font face **FF_SYSTEM**.

DeskMate Technical Reference

Font Resource

```
int font_face_unlock ( fpFONT_FACE )  
FONT_FACEfar    *fpFONT_FACE;
```

font_face_unlock allows the caller to unlock a particular font face that was previously locked by the call to **font_face_lock**. The default state of a font face is unlocked.

Input Parameters

fpFONT_FACE is a far pointer to a **FONT_FACE** data structure. The contents should be the same as the one passed for the **font_face_lock**.

Return Value

DMF_OK (0) indicates success in unlocking the font face.
DMF_ERROR (-1) indicates unsuccessful in locking the font face.

DeskMate Technical Reference Font Resource

```
BITMAP far *font_get_bitmap ( fpFONT_FACE, Character )  
FONT_FACE      far      *fpFONT_FACE;  
unsigned int      Character;
```

font_get_bitmap returns a far pointer to a data structure of type **BITMAP**, which contains dimensions of the bitmap and a far pointer to bitmap for the specified character **Character** for the font.

Input Parameters

fpFONT_FACE is a far pointer to a **FONT_FACE** data structure.
Character contains the ASCII value of the character for which a bitmap is required.

Return Value

A far pointer to a **BITMAP** data structure is returned providing the necessary information about the generated character.

If the character bitmap is too large to be cached then the relevant dimensional information is returned within the far pointer to a **BITMAP** data structure, the far pointer to the bitmapped character element **fpBitmap** is set to **ZERO**.

The far pointer to the character element **fpBitmap** is set to **DMF_ERROR (-1)** if a total failure to generate a bitmap occurred.

Special Notes

The element **fpBitmap** in the returned **BITMAP** structure points to a buffer in which to store the bitmap. The length of this buffer should NOT be less than $((\text{pixel_width}/8) * \text{pixel_height})$ or the buffer will be overrun. The **pixel_width** and **pixel_height** can be smaller than the actual character, therefore asking for a particular section of a character at a time. The bitmap is in the format of **640 * 350 * 2** color, where 0 = white and 1 = black.

DeskMate Technical Reference Font Resource

```
int font_get_cell ( fpFONT_FACE, fpFONT_CELL )  
FONT_FACE far    *fpFONT_FACE;  
FONT_CELL far    *fpFONT_CELL;
```

font_get_cell provides the basic font character dimensions. By default, the function returns the font height and leading in the far pointer **fpFONT_CELL** data structure **FONT_CELL**. **font_get_cell** also returns the character bitmap rendering position in relation to a baseline. The secondary information is only updated and returned if a character was specified in the far pointer **fpFONT_CELL** data structure. (**fpFONT_CELL->ascii_ch** element of the **FONT_CELL** data structure.)

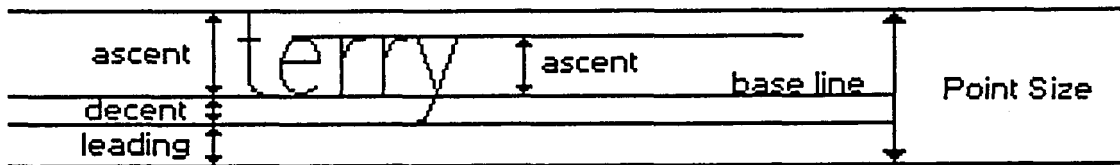
Input Parameters

fpFONT_FACE is a far pointer to a **FONT_FACE** data structure.
fpFONT_CELL is far pointer to a **FONT_CELL** data structure which is updated on return from the call. The **ascii_ch** element can be set to the character.

Return Value

fpFONT_CELL is updated with the appropriate dimension data about the font.
DMF_OK (0) is returned indicating a success.
DMF_ERROR (-1) indicating a failure.

Special Notes



DeskMate Technical Reference

Font Resource

```
void font_get_dpi ( Horizontal, Vertical )  
unsigned int far *Horizontal;  
unsigned int far *Vertical;
```

font_get_dpi returns the current effective resolution. The horizontal and vertical resolution in dots per inch is returned as used by the font engine to generate the bitmap.

Input Parameters

Far pointers to **Horizontal** and **Vertical** are provided.

Return Value

The two pointers to **Horizontal** and **Vertical** are updated with the current resolution in dots per inch.

DeskMate Technical Reference

Font Resource

```
int font_get_face ( fpFACE, fpFONT_FACE )  
FACE             far    *fpFACE;  
FONT_FACEfar     *fpFONT_FACE;
```

font_get_face is made by applications who wish to get a particular font face from a list of font faces. **font_get_face** provides applications, which do not have the space to store the font faces, a way to keep only the index number to a particular font.

Enumerated calls are made to the **font_get_face** to retrieve all the font face information for the particular font face type specified. The information is updated into the buffer provided by the far pointer to **FONT_FACE** structure.

font_face_support returns the number of fonts available. The application must call **font_get_face** that many times to get the information. This information is necessary for the font engine to return a bitmap.

Input Parameters

fpFACE is a far pointer to a **FACE** structure.

fpFONT_FACE is a far pointer to a **FONT_FACE** structure.

The **FACE** structure contains the font face type requested by the application for the particular font face in the list. The element **request** in the **FACE** structure indicates the starting position within the list and the element is incremented every time **font_get_face** is called.

Return Value

DMF_OK (0) if a font face was supplied.

DMF_ERROR (-1) if a font face was not found.

If successful the characteristics of the new font are updated into the data structure pointed by **fpFONT_FACE**. The **request** element in the **FACE** structure is incremented to the next font face.

Special Notes

Request element data contains the read/update defines when a call is made to **font_face_support** but an integer index is kept for function call **font_get_face**.

DeskMate Technical Reference

Font Resource

```
int font_get_style ( fpFONT_FACE, fpOUTPUT_STYLE_INFO )  
FONT_FACE          far    *fpFONT_FACE;  
OUTPUT_STYLE_INFO  far    *fpOUTPUT_STYLE_INFO;
```

font_get_style returns the output style positional information as required to support such attributes as underline, superscript and subscript for each types of font. These attributes are not generated as part of the character bitmap and therefore these values appended separately.

Input Parameters

fpFONT_FACE is a far pointer to a **FONT_FACE** data structure.

fpOUTPUT_STYLE_INFO is a far pointer to an **OUTPUT_STYLE_INFO** data structure.

Return Value

The **OUTPUT_STYLE_INFO** data structure is updated.

DMF_OK (0) is returned indicating a success.

DMF_ERROR (-1) indicating a failure.

DeskMate Technical Reference

Font Resource

```
unsigned int font_get_width ( fpFONT_FACE, fpString, nChar )  
FONT_FACEfar    *fpFontFaceInfo;  
char            far    *fpString;  
int             nChar;
```

font_get_width returns the width of a string for the specified font.

Input Parameters

fpFONT_FACE is a far pointer to a **FONT_FACE** data structure.

fpString is far pointer to an ASCII string.

nChar indicates the length of the string to calculate.

Return Value

An integer representing the width of the string is returned in DeskMate world co-ordinate units.

A zero (0) is returned as an indication of failure.

Special Notes

The far pointer **fpString** may also contain embedded character attributes. The sequence of bytes that determines if a an attribute follows is; hexadecimal value 0x1B followed by unsigned long **bStyle** which indicates the fonts style required.

DeskMate Technical Reference

Font Resource

```
int font_plot_bitmap ( fpFONT_FACE, Character, fpBITMAP )
FONT_FACE      far    *fpFONT_FACE;
unsigned int    Character;
BITMAP         far    *fpBITMAP;
```

font_plot_bitmap plots a partial or an entire character bitmap directly into the buffer provided. The far pointer to the buffer is provided within in the data structure **BITMAP** pointed to by **fpBITMAP**.

Input Parameters

fpFONT_FACE is a far pointer to a **FONT_FACE** data structure which describes the font.
Character contains the ASCII value of the character for which a bitmap is required.
fpBITMAP is far pointer to a **BITMAP** structure as returned by **font_get_bitmap** except a far pointer to the bitmap character is set to your buffer.

Return Value

The far pointer to your buffer **fpBITMAP**→**fpBitmap** is updated with the character bitmap generated for the region requested.

DMF_OK (0) is returned indicating a success.

DMF_ERROR (-1) indicating an error in generating the character.

Special Notes

The element **fpBitmap** in the returned **BITMAP** structure points to a buffer in which to store the bitmap. The length of this buffer should NOT be less than $((\text{pixel_width}/8) * \text{pixel_height})$ or the buffer will be overrun. The **pixel_width** and **pixel_height** can be smaller than the actual character, therefore asking for a particular section of a character at a time. The bitmap is in the format of 640 * 350 * 2 color, where 0 = white and 1 = black.

DeskMate Technical Reference

Font Resource

int **font_set_buffer** (CacheSize)
UNSIGNED INT CacheSize ;

font_set_buffer provides a mechanism for application programs to request the font resource to allocate memory buffers for caching, generated bitmap characters and width tables of each character of each faces.

Input Parameters

CacheSize is an integer representing the number of paragraphs of memory required. A paragraph is 16 bytes.

Return Value

DMF_OK (0) indicates success in allocating the cache buffer from DOS.

An integer (Non-Zero) indicates a failure in allocating a buffer, but indicates the amount of paragraphs available.

Special Notes

CacheSize could be set to DMF_FREE to free the previous allocated cache buffer to zero. This should be done prior to running accessories. This is done automatically prior to task switching.

DeskMate Technical Reference

Font Resource

```
void font_set_dpi ( Horizontal, Vertical )  
unsigned int Horizontal;  
unsigned int Vertical;
```

font_set_dpi sets the effective resolution for font generation. Sets the effective horizontal and vertical resolution for font bitmap generation for different output devices.

Input Parameters

Horizontal and Vertical specifies the horizontal and vertical resolution in dots per inch.

Return Value

None.

Screen Save Resource

Table of Contents

General Description/Notes	28- 1
Structures/Defines	28- 2
ssm_bind_end	Terminates the Screen Save Resource.....28- 3
ssm_bind_init	Initailaizes the Screen Save Resource...28- 4
ssm_get_size	Gets the size of the buffer needed.....28- 5
ssm_restore_screen	Restores the screen.....28- 6
ssm_save_screen	Stores screen data in compressed form...28- 7

DeskMate Technical Reference Screen Save Resource

General Description/Notes

The use of this resource requires the application to link with the DeskMate 03.05 and later versions of the DeskMate libraries, DM.LIB and DMMED.LIB.

The Screen Save Resource allows an application to save and restore an area of the screen. The screen image to save is compressed using one of two application defined compression algorithms. The Screen Save Manager Resource (SSM) provides routines for high speed or high efficiency compression. The SSM will typically compress the screen image to 25 percent of its original size leaving more memory for use by the application.

The routines in this section are used by the two calls `vid_get_screen` and `vid_restore_screen`.

The busy icon is turned on by the SSM upon completion of any of the SSM routines. If the calling application desires the busy icon to remain disabled, the application should disable the busy icon after all requests to SSM functions.

DeskMate Technical Reference Screen Save Resource

Structures/Defines

A structure is passed to the SSM functions. This structure is defined as follows:

```
struct      ssm_defn
{
    char      mode;          /* HI_SPEED or HI_COMPRESS */
    int       xorg;          /* origin and extent of screen area */
    int       yorg;          /* area to capture */
    int       xext;
    int       yext;
    int       buffer_seg;    /* Applications data segment */
    char      *buffer;       /* pointer to buffer to contain data */
};
typedef struct ssm_defn SCREEN_SAVE;
```

mode defines which algorithms to use. This element should be set to HI_SPEED or HI_COMPRESSION.

xorg defines the X origin of the screen image to read or write.

yorg defines the Y origin of the screen image to read or write.

xorg defines the X extent of the screen image to read or write.

yorg defines the Y extent of the screen image to read or write.

buffer_seg is the application data segment.

buffer is a pointer to a buffer to contain the compressed screen image data.

DeskMate Technical Reference

Screen Save Resource

ssm_bind_end ()

ssm_bind_end terminates the use of the SSM resource and tells the Desk Executive to unload the resource if the memory space is needed.

Input Parameters

None.

Return Value

None.

Special Notes

No SSM calls may be made after terminating the binding by calling **ssm_bind_end**.

DeskMate Technical Reference

Screen Save Resource

ssm_bind_init ()

ssm_bind_init loads the SSM resource and binds the application to the resource which allows the application to utilize the SSM calls.

Input Parameters

None.

Return Value

Zero if the Screen Save Resource loaded successfully.
DM_ERROR if the Screen Save Resource failed to load.

Special Notes

ssm_bind_init must be called before using any of the SSM function calls. The busy icon is enabled upon returning to the application from any of the SSM function calls. If the application desires the busy icon to be disabled, the application should call **vid_busy_disable** to disable the busy icon.

DeskMate Technical Reference

Screen Save Resource

```
long ssm_get_size( pScreenSave)  
SCREEN_SAVE *pScreenSave;
```

ssm_get_size returns the size of buffer needed to contain the compressed data for the specified screen area.

Input Parameters

pScreenSave points to SCREEN_SAVE structure.

Return Value

<long>

The required size is returned as a long.

Special Notes

ssm_bind_init must be called to load the SSM resource prior to using **ssm_get_size**.

DeskMate Technical Reference

Screen Save Resource

ssm_restore_screen (pScreenSave)
SCREEN_SAVE *pScreenSave;

ssm_restore_screen restores the screen using the data retrieved by **ssm_save_screen**.

Input Parameters

pScreenSave points to SCREEN_SAVE structure.

Return Value

None.

Special Notes

ssm_bind_init must be called to load the SSM resource prior to using **ssm_restore_screen**.

DeskMate Technical Reference

Screen Save Resource

ssm_save_screen(pScreenSave)
SCREEN_SAVE *pScreenSave;

ssm_save_screen reads an area of the screen and stores the screen data in compressed form.

Input Parameters

pScreenSave points to SCREEN_SAVE structure.

Return Value

None.

Special Notes

ssm_bind_init must be called to load the SSM resource prior to using **ssm_save_screen**.