
OpenVMS/Hanyu RTL Chinese Screen Management (SMG\$) Manual

Order Number: AA-PXHDB-TE

August 2005

This manual documents the Chinese screen management routines contained in the SMG\$ facility of the OpenVMS Run-Time Library.

Revision/Update Information: This document supersedes the VMS RTL Chinese SMG\$ Manual Version 5.5.

Software Version: OpenVMS/Hanyu Alpha Version 8.2

**Hewlett-Packard Company
Palo Alto, California**

© Copyright 2005 Hewlett-Packard Development Company, L.P.

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Printed in Singapore.

ZK5935

This document was prepared using DECdocument, Version 3.3-1b.

Contents

Preface	xiii
1 Overview of the Chinese Screen Management Facility (SMG\$)	
1.1 Pasteboards	1-4
1.2 Virtual Displays	1-5
1.3 Viewports	1-7
1.4 Virtual Keyboards	1-7
2 Chinese Screen Management Output Operations	
2.1 Composition Operations	2-1
2.1.1 Paste Operation	2-1
2.1.2 Unpaste Operation	2-2
2.1.3 Repaste Operation	2-3
2.1.4 Move Operation	2-3
2.1.5 Delete and Pop Operations	2-4
2.1.6 Occlusion Check Operation	2-5
2.2 Output Through Virtual Displays	2-6
2.2.1 Cursor Position	2-7
2.2.2 Deletion Operations	2-8
2.2.3 Erasure Operations	2-8
2.2.4 Insertion Operations	2-8
2.2.5 Writing Operations	2-9
2.2.5.1 Character-Oriented Output	2-9
2.2.5.2 Line-Oriented Output	2-9
2.2.6 Changing the Rendition of a Virtual Display	2-10
2.2.7 Drawing and Removing Drawn Lines and Characters	2-11
2.2.8 Displaying External Text	2-12
2.2.9 Reading from a Virtual Display	2-12
2.2.10 Viewports	2-12
2.2.10.1 Creating a Viewport	2-13
2.2.10.2 Deleting a Viewport	2-13
2.2.10.3 Pasting and Unpasting a Viewport	2-13
2.2.10.4 Scrolling and Moving a Viewport	2-13
2.2.10.5 Changing Viewport Characteristics	2-14
2.2.11 Menus	2-14
2.2.11.1 Creating a Menu	2-14
2.2.11.2 Deleting a Menu	2-14
2.2.11.3 Selecting from a Menu	2-15
2.2.11.3.1 Default Mode	2-15
2.2.11.3.2 RETURN_IMMED Mode	2-15
2.2.11.3.3 REMOVE_ITEM Mode	2-15
2.2.12 Saving a Virtual Display	2-15

2.2.13	Changing Terminal Characteristics	2-15
2.2.14	Setting the Default Character Set of a Virtual Display	2-16
2.3	Operational Controls	2-16
2.3.1	Minimal Update	2-16
2.3.2	Buffering	2-16
2.3.3	Tabs	2-17
2.4	Batching Output Operations	2-17
2.4.1	Display Update Batching	2-17
2.4.2	Pasteboard Update Batching	2-17

3 Chinese Screen Management Input Operations

3.1	Obtaining Data from Virtual Keyboards	3-1
3.2	Setting and Retrieving Virtual Keyboard Characteristics	3-2
3.3	Line Composition Using Keypad Keys	3-2
3.4	States	3-3
3.5	Terminators	3-3
3.6	Line Recall and the Recall Buffer	3-6
3.7	Interaction of Input and Output	3-7

4 Advanced Screen Management Features

4.1	Asynchronous Events	4-1
4.1.1	Broadcast Messages	4-2
4.1.2	Unsolicited Input	4-2
4.1.3	Out-of-Band ASTs	4-2
4.2	Subprocesses	4-2
4.2.1	Creating a Subprocess	4-2
4.2.2	Deleting a Subprocess	4-2
4.2.3	Executing Commands in a Subprocess	4-3
4.3	Moving the Cursor	4-3
4.4	Exit Handler	4-3

5 Support for Third Party Terminals

5.1	TERMTABLE Interface Routines	5-1
5.2	Capability Fields	5-3
5.2.1	Boolean Capability Fields	5-4
5.2.2	Numeric Capability Fields	5-5
5.2.3	String Capability Fields	5-6
5.2.4	Argument Substitution	5-14
5.2.5	Arithmetic Operations	5-15
5.3	Creating an OpenVMS Terminal Capabilities File	5-16
5.4	Examples	5-16
5.5	Creating TERMTABLE.EXE	5-20
5.6	Capability Fields Used by Chinese Screen Management	5-21
5.7	Input Support for Foreign Terminals	5-21
5.8	Support for SET and SHOW TERMINAL Commands	5-22

6 Using Chinese Screen Management Routines to Develop New Programs

6.1	Calling Routines That Do Not Use the Chinese Screen Management Facility	6-3
-----	---	-----

7 Examples of Calling SMG\$ Routines

SMG\$ Reference Section

SMG\$ADD_KEY_DEF	SMG-3
SMG\$BEGIN_DISPLAY_UPDATE	SMG-7
SMG\$BEGIN_PASTEBOARD_UPDATE	SMG-8
SMG\$CANCEL_INPUT	SMG-9
SMG\$CHANGE_PBD_CHARACTERISTICS	SMG-10
SMG\$CHANGE_RENDITION	SMG-13
SMG\$CHANGE_VIEWPORT	SMG-17
SMG\$CHANGE_VIRTUAL_DISPLAY	SMG-23
SMG\$CHECK_FOR_OCCLUSION	SMG-26
SMG\$CONTROL_MODE	SMG-30
SMG\$COPY_VIRTUAL_DISPLAY	SMG-33
SMG\$CREATE_KEY_TABLE	SMG-39
SMG\$CREATE_MENU	SMG-40
SMG\$CREATE_PASTEBOARD	SMG-46
SMG\$CREATE_SUBPROCESS	SMG-50
SMG\$CREATE_VIEWPORT	SMG-54
SMG\$CREATE_VIRTUAL_DISPLAY	SMG-58
SMG\$CREATE_VIRTUAL_KEYBOARD	SMG-63
SMG\$CURSOR_COLUMN	SMG-67
SMG\$CURSOR_ROW	SMG-68
SMG\$DEFINE_KEY	SMG-69
SMG\$DEL_TERM_TABLE	SMG-71
SMG\$DELETE_CHARS	SMG-72
SMG\$DELETE_KEY_DEF	SMG-77
SMG\$DELETE_LINE	SMG-79
SMG\$DELETE_MENU	SMG-83
SMG\$DELETE_PASTEBOARD	SMG-84
SMG\$DELETE_SUBPROCESS	SMG-86
SMG\$DELETE_VIEWPORT	SMG-87
SMG\$DELETE_VIRTUAL_DISPLAY	SMG-88
SMG\$DELETE_VIRTUAL_KEYBOARD	SMG-89
SMG\$DISABLE_BROADCAST_TRAPPING	SMG-90
SMG\$DISABLE_UNSOLICITED_INPUT	SMG-100
SMG\$DRAW_CHAR	SMG-102
SMG\$DRAW_LINE	SMG-106
SMG\$DRAW_RECTANGLE	SMG-112
SMG\$ENABLE_UNSOLICITED_INPUT	SMG-117
SMG\$END_DISPLAY_UPDATE	SMG-120

SMG\$END_PASTEBOARD_UPDATE	SMG-121
SMG\$ERASE_CHARS	SMG-122
SMG\$ERASE_COLUMN	SMG-127
SMG\$ERASE_DISPLAY	SMG-129
SMG\$ERASE_LINE	SMG-134
SMG\$ERASE_PASTEBOARD	SMG-138
SMG\$EXECUTE_COMMAND	SMG-140
SMG\$FIND_CURSOR_DISPLAY	SMG-142
SMG\$FLUSH_BUFFER	SMG-144
SMG\$GET_BROADCAST_MESSAGE	SMG-145
SMG\$GET_DISPLAY_ATTR	SMG-147
SMG\$GET_KEY_DEF	SMG-150
SMG\$GET_KEYBOARD_ATTRIBUTES	SMG-153
SMG\$GET_NUMERIC_DATA	SMG-156
SMG\$GET_PASTEBOARD_ATTRIBUTES	SMG-158
SMG\$GET_PASTING_INFO	SMG-162
SMG\$GET_TERM_DATA	SMG-164
SMG\$GET_VIEWPORT_CHAR	SMG-166
SMG\$HOME_CURSOR	SMG-170
SMG\$INIT_TERM_TABLE	SMG-172
SMG\$INIT_TERM_TABLE_BY_TYPE	SMG-174
SMG\$INSERT_CHARS	SMG-176
SMG\$INSERT_LINE	SMG-183
SMG\$INVALIDATE_DISPLAY	SMG-190
SMG\$KEYCODE_TO_NAME	SMG-191
SMG\$LABEL_BORDER	SMG-193
SMG\$LIST_KEY_DEFS	SMG-199
SMG\$LIST_PASTEBOARD_ORDER	SMG-202
SMG\$LIST_PASTING_ORDER	SMG-204
SMG\$LOAD_KEY_DEFS	SMG-206
SMG\$LOAD_VIRTUAL_DISPLAY	SMG-208
SMG\$MOVE_TEXT	SMG-210
SMG\$MOVE_VIRTUAL_DISPLAY	SMG-214
SMG\$NAME_TO_KEYCODE	SMG-216
SMG\$PASTE_VIRTUAL_DISPLAY	SMG-218
SMG\$POP_VIRTUAL_DISPLAY	SMG-221
SMG\$PUT_CHARS	SMG-223
SMG\$PUT_CHARS_HIGHWIDE	SMG-228
SMG\$PUT_CHARS_MULTI	SMG-232
SMG\$PUT_CHARS_WIDE	SMG-237
SMG\$PUT_HELP_TEXT	SMG-241
SMG\$PUT_LINE	SMG-245
SMG\$PUT_LINE_HIGHWIDE	SMG-252
SMG\$PUT_LINE_MULTI	SMG-256
SMG\$PUT_LINE_WIDE	SMG-261
SMG\$PUT_STATUS_LINE	SMG-266
SMG\$READ_COMPOSED_LINE	SMG-268

SMG\$READ_FROM_DISPLAY	SMG-274
SMG\$READ_KEYSTROKE	SMG-279
SMG\$READ_STRING	SMG-287
SMG\$REMOVE_LINE	SMG-298
SMG\$REPAINT_SCREEN	SMG-301
SMG\$REPASTE_VIRTUAL_DISPLAY	SMG-305
SMG\$REPLACE_INPUT_LINE	SMG-311
SMG\$RESTORE_PHYSICAL_SCREEN	SMG-315
SMG\$RETURN_CURSOR_POS	SMG-317
SMG\$RETURN_INPUT_LINE	SMG-319
SMG\$RING_BELL	SMG-323
SMG\$SAVE_PHYSICAL_SCREEN	SMG-324
SMG\$SAVE_VIRTUAL_DISPLAY	SMG-326
SMG\$SCROLL_DISPLAY_AREA	SMG-328
SMG\$SCROLL_VIEWPORT	SMG-331
SMG\$SELECT_FROM_MENU	SMG-338
SMG\$SET_BROADCAST_TRAPPING	SMG-356
SMG\$SET_CURSOR_ABS	SMG-358
SMG\$SET_CURSOR_MODE	SMG-360
SMG\$SET_CURSOR_REL	SMG-362
SMG\$SET_DEFAULT_STATE	SMG-364
SMG\$SET_DISPLAY_SCROLL_REGION	SMG-366
SMG\$SET_KEYPAD_MODE	SMG-368
SMG\$SET_OUT_OF_BAND_ASTS	SMG-370
SMG\$SET_PHYSICAL_CURSOR	SMG-374
SMG\$SET_TERM_CHARACTERISTICS	SMG-376
SMG\$UNPASTE_VIRTUAL_DISPLAY	SMG-379

Index

Examples

7-1	Using SMG\$ Routines in VAX Ada	7-1
7-2	Using SMG\$ Routines in VAX BASIC	7-4
7-3	Using SMG\$ Routines in VAX BLISS32	7-6
7-4	Using SMG\$ Routines in VAX COBOL	7-9
7-5	Using SMG\$ Routines in DEC Fortran	7-11
7-6	Using SMG\$ Routines in VAX MACRO	7-13
7-7	Using SMG\$ Routines in VAX Pascal	7-16
7-8	Using SMG\$ Routines in VAX PL/I	7-18
7-9	Using SMG\$ Routines in VAX RPG II	7-21

Figures

1-1	Pasteboard Coordinate System	1-5
2-1	Paste Operation	2-2
2-2	Unpaste Operation	2-2
2-3	Repaste Operation	2-3
2-4	Move Operation	2-4
2-5	Pop Operation	2-5
2-6	Occlusion Check	2-6
2-7	Lines Drawn Without SMG\$DRAW_LINE	2-12
2-8	Lines Drawn with SMG\$DRAW_LINE	2-12
SMG-1	Output Generated After Virtual Displays Are Pasted	SMG-20
SMG-2	Output Generated After the Viewport Is Created	SMG-21
SMG-3	Output Generated After Calling SMG\$CHANGE_VIEWPORT	SMG-22
SMG-4	Output Generated by FORTRAN Program Calling SMG\$CHECK_FOR_OCCLUSION	SMG-29
SMG-5	First Virtual Display Generated by This Example	SMG-36
SMG-6	Second Virtual Display Generated by This Example	SMG-37
SMG-7	Output Generated After the Call to SMG\$COPY_VIRTUAL_DISPLAY	SMG-38
SMG-8	Output Generated by Creating a Viewport	SMG-57
SMG-9	Output of FORTRAN Program Calling SMG\$CREATE_VIRTUAL_DISPLAY	SMG-62
SMG-10	Output Generated Before the Call to SMG\$DELETE_CHARS	SMG-75
SMG-11	Output Generated After the Call to SMG\$DELETE_CHARS	SMG-76
SMG-12	Output Generated by FORTRAN Program Before the Call to SMG\$DELETE_LINE	SMG-81
SMG-13	Output Generated After the Call to SMG\$DELETE_LINE	SMG-82
SMG-14	Output Generated Before Any Input or Messages Are Trapped	SMG-97
SMG-15	Output Generated After a Broadcast Message Is Trapped	SMG-98
SMG-16	Output Generated After a Call to LIB\$SPAWN	SMG-99
SMG-17	Output Generated by FORTRAN Program Calling SMG\$DRAW_LINE	SMG-111
SMG-18	Output Generated by FORTRAN Program Calling SMG\$DRAW_RECTANGLE	SMG-116
SMG-19	AST Routine Arguments	SMG-118
SMG-20	Output Before the Call to SMG\$ERASE_CHARS	SMG-125
SMG-21	Output After the Call to SMG\$ERASE_CHARS	SMG-126
SMG-22	Initial Output of FORTRAN Program Calling SMG\$ERASE_DISPLAY	SMG-132
SMG-23	Output Displayed After the Call to SMG\$ERASE_DISPLAY	SMG-133
SMG-24	Initial Output Generated by FORTRAN Program Calling SMG\$ERASE_LINE	SMG-136
SMG-25	Output Generated After the Call to SMG\$ERASE_LINE	SMG-137
SMG-26	Output Generated by SMG\$GET_VIEWPORT_CHAR	SMG-169
SMG-27	Output Generated by FORTRAN Program Before the Call to SMG\$INSERT_CHARS	SMG-181
SMG-28	Output Generated by FORTRAN Program After the Call to SMG\$INSERT_CHARS	SMG-182

SMG-29	Output Generated Before the Call to SMG\$INSERT_LINE	SMG-188
SMG-30	Output Generated After the Call to SMG\$INSERT_LINE	SMG-189
SMG-31	Output Generated by Program Calling SMG\$LABEL_BORDER	SMG-198
SMG-32	Arguments for Moving Text	SMG-213
SMG-33	Output of FORTRAN Program Calling SMG\$PUT_CHARS	SMG-227
SMG-34	Output Generated by FORTRAN Program Calling SMG\$PUT_LINE	SMG-250
SMG-35	Output Generated by FORTRAN Program Calling SMG\$PUT_LINE_WIDE	SMG-265
SMG-36	Output Generated by FORTRAN Program Calling SMG\$READ_FROM_DISPLAY	SMG-278
SMG-37	Output Generated by FORTRAN Program Calling SMG\$READ_KEYSTROKE	SMG-285
SMG-38	Output Generated Before the Call to SMG\$READ_STRING	SMG-296
SMG-39	Output Generated After the Call to SMG\$READ_STRING	SMG-297
SMG-40	Output Generated by FORTRAN Program Calling SMG\$REPAINT_SCREEN	SMG-303
SMG-41	Output Generated by FORTRAN Program Calling SMG\$REPAINT_SCREEN	SMG-304
SMG-42	Output Before the Call to SMG\$MOVE_VIRTUAL_DISPLAY	SMG-308
SMG-43	Output Displayed After the Call to SMG\$MOVE_VIRTUAL_DISPLAY	SMG-309
SMG-44	Output Displayed After the Call to SMG\$REPASTE_VIRTUAL_DISPLAY	SMG-310
SMG-45	Output Generated by Pasting the Virtual Display	SMG-334
SMG-46	Output Generated After First Call to SMG\$SCROLL_VIEWPORT	SMG-335
SMG-47	Output Generated After the Second Call to SMG\$SCROLL_VIEWPORT	SMG-336
SMG-48	Output Generated After the Last Call to SMG\$SCROLL_VIEWPORT	SMG-337
SMG-49	Output Generated Before a Menu Selection Is Made	SMG-345
SMG-50	Output Generated After the User Selects an Item	SMG-346
SMG-51	Output Generated Before the User Selects a Menu Item	SMG-349
SMG-52	Output Generated After Two Selections	SMG-350
SMG-53	Output Generated After EXIT Is Selected	SMG-351
SMG-54	Output Generated Before a Menu Item Is Selected	SMG-353
SMG-55	Output Generated After Three Menu Selections	SMG-354
SMG-56	Output Generated After Program Completion	SMG-355

Tables

1	Character set supported by Chinese terminal	xvi
1-1	Routines that Support Third Party Terminals	1-1
1-2	Input Routines	1-1
1-3	Output Routines	1-2
2-1	Default Character Set Names	2-16
3-1	Terminator Values	3-4
5-1	Boolean Capabilities	5-4

5-2	Numeric Capabilities	5-6
5-3	String Capabilities	5-7
5-4	String_2 Capabilities	5-12

Preface

This manual provides users of the HP OpenVMS operating system with detailed usage and reference information on the Chinese screen management routines supplied in the SMG\$ facility of the Run-Time Library (RTL).

Intended Audience

This manual is intended for system and application programmers who want to call the Run-Time Library routines.

Document Structure

This manual is organized into two parts as follows:

- The introductory chapters provide guidelines and reference material on specific types of Chinese screen management routines. The material is covered as follows:
 - Chapter 1 lists the SMG\$ routines and provides a brief overview of the major SMG\$ components.
 - Chapter 2 discusses output operations provided by the Chinese Screen Management Facility.
 - Chapter 3 describes Chinese screen management routines used to perform input from a virtual keyboard.
 - Chapter 4 discusses the Chinese Screen Management Facility's advanced features.
 - Chapter 5 discusses a method of supporting foreign terminals.
 - Chapter 6 discusses some recommended methods for using the Chinese Screen Management Facility for developing new programs.
 - Chapter 7 contains examples demonstrating how to call some SMG\$ routines from major OpenVMS languages.
- Part II provides detailed reference information on each routine contained in the SMG\$ facility of the Run-Time Library. This information is presented using the documentation format described in *OpenVMS Programming Interfaces: Calling a System Routine*. Routine descriptions appear in alphabetical order by routine name.

Associated Documents

The Run-Time Library routines are documented in a series of reference manuals. A description of how the Run-Time Library routines are accessed is presented in *OpenVMS Programming Interfaces: Calling a System Routine*. A description of OpenVMS features and functionality available through calls to the SMG\$ Run-Time Library appears in *OpenVMS Programming Concepts Manual*. Descriptions of the other RTL facilities and their corresponding routines and usages are discussed in the following books:

- *DPML, Digital Portable Mathematics Library*
- *OpenVMS RTL DECtalk (DTK\$) Manual*
- *OpenVMS RTL Library (LIB\$) Manual*
- *OpenVMS RTL Mathematics (MTH\$) Manual*
- *OpenVMS RTL General Purpose (OTS\$) Manual*
- *OpenVMS RTL Parallel Processing (PPL\$) Manual*
- *OpenVMS RTL Screen Management (SMG\$) Manual*
- *OpenVMS RTL String Manipulation (STR\$) Manual*

Guide to DECthreads contains guidelines and reference information for DECthreads, Digital's Multithreading Run-Time Library.

Application programmers using any programming language can refer to *Guide to Creating OpenVMS Modular Procedures* for writing modular and reentrant code.

High-level language programmers will find additional information on calling Run-Time Library routines in their language reference manual. Additional information may also be found in the language user's guide provided with your OpenVMS language software.

For a complete list and description of the manuals in the OpenVMS documentation set, see *Overview of OpenVMS Documentation*.

Conventions

In this manual, "OpenVMS I64" means the HP OpenVMS I64 operating system, "OpenVMS Alpha" means the HP OpenVMS Alpha operating system, "OpenVMS VAX" means the HP OpenVMS VAX operating system, and "OpenVMS" means the HP OpenVMS I64 operating system, the HP OpenVMS Alpha operating system and the HP OpenVMS VAX operating system.

The following conventions are used to identify information specific to OpenVMS Alpha or to OpenVMS VAX:



The Alpha icon denotes the beginning of information specific to OpenVMS Alpha.



The VAX icon denotes the beginning of information specific to OpenVMS VAX.



The diamond symbol denotes the end of a section of information specific to OpenVMS Alpha or to OpenVMS VAX.

The following conventions are used in this manual:

Ctrl/x	A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 x	A sequence such as PF1 x indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.
GOLD x	A sequence such as GOLD x indicates that you must first press and release the key defined GOLD, then press and release another key. GOLD key sequences can also have a slash (/), dash (-), or underscore (_) as a delimiter in EVE commands.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
...	A horizontal ellipsis in examples indicates one of the following possibilities: <ul style="list-style-type: none">• Additional optional arguments in a statement have been omitted.• The preceding item or items can be repeated one or more times.• Additional parameters, values, or other information can be entered.
.	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
()	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[]	In format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification, or in the syntax of a substring specification in an assignment statement.)
{ }	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in Bookreader versions of the manual.
<i>italic text</i>	Italic text emphasizes important information, indicates variables, and indicates complete titles of manuals. Italic text also represents information that can vary in system messages (for example, Internal error <i>number</i>), command lines (for example, /PRODUCER= <i>name</i>), and command parameters in text.
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, the name of a file protection code, or the abbreviation for a system privilege.

- A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.

numbers All numbers in the text are assumed to be decimal, unless otherwise noted. Nondecimal radices—binary, octal, or hexadecimal—are explicitly indicated.

Other conventions used in the documentation of Run-Time Library routines are described in the *OpenVMS Programming Interfaces: Calling a System Routine*.

Hardware

Chinese Screen Management Facility supports the following terminals:

- VT382-D

Table 1 shows the character set which each terminal supports.

Table 1 Character set supported by Chinese terminal

Character Set	VT382-D
ASCII	Yes
DEC Hanyu	Yes
DEC Special Graphics	Yes
DEC Supplemental	Yes
ISO Latin-1	Yes

Overview of the Chinese Screen Management Facility (SMG\$)

This manual discusses the Run-Time Library routines that perform terminal-independent functions. The most important aspect of the Chinese Screen Management Facility is that user programs are entirely separated from the physical devices that actually perform input and output. Instead of writing directly to a physical screen, the user program writes to a **virtual display**. Similarly, instead of performing input directly from a physical keyboard, user programs perform input from a **virtual keyboard**. (Virtual displays and virtual keyboards are logical entities whose usage is described more fully in the following sections.) This separation of virtual operations from physical operations is what allows input/output to be terminal independent.

The SMG\$ routines listed below assist you in designing, composing, and keeping track of complex images on a video screen. These routines are meant for the types of operations you would normally perform on a VT100-class terminal; they also provide software emulation of screen management functions on terminals that do not have these functions implemented in their hardware. The following lists contain all the Chinese screen management routines grouped according to their functions.

Table 1–1 lists routines that support third party terminals; Chapter 5 discusses the routines.

Table 1–2 lists the SMG\$ input routines; Chapter 3 discusses the routines.

Table 1–3 lists the SMG\$ output routines; Chapter 2 discusses the routines.

Table 1–1 Routines that Support Third Party Terminals

Routines that Support Third Party Terminals	
SMG\$DEL_TERM_TABLE	SMG\$GET_NUMERIC_DATA
SMG\$GET_TERM_DATA	SMG\$INIT_TERM_TABLE
SMG\$INIT_TERM_TABLE_BY_TYPE	

Table 1–2 Input Routines

Input Routines	
SMG\$ADD_KEY_DEF	SMG\$CANCEL_INPUT
SMG\$CREATE_KEY_TABLE	SMG\$CREATE_VIRTUAL_KEYBOARD
SMG\$DEFINE_KEY	SMG\$DELETE_KEY_DEF

(continued on next page)

Overview of the Chinese Screen Management Facility (SMG\$)

Table 1–2 (Cont.) Input Routines

Input Routines	
SMG\$DELETE_VIRTUAL_KEYBOARD	SMG\$GET_KEY_DEF
SMG\$GET_KEYBOARD_ATTRIBUTES	SMG\$KEYCODE_TO_NAME
SMG\$LIST_KEY_DEFS	SMG\$LOAD_KEY_DEFS
SMG\$NAME_TO_KEYCODE	SMG\$READ_COMPOSED_LINE
SMG\$READ_KEYSTROKE	SMG\$READ_STRING
SMG\$REPLACE_INPUT_LINE	SMG\$RETURN_INPUT_LINE
SMG\$SET_DEFAULT_STATE	SMG\$SET_KEYPAD_MODE

Table 1–3 Output Routines

Output Routines	
SMG\$BEGIN_DISPLAY_UPDATE	SMG\$BEGIN_PASTEBOARD_UPDATE
SMG\$CHANGE_PBD_CHARACTERISTICS	SMG\$CHANGE_RENDITION
SMG\$CHANGE_VIEWPORT	SMG\$CHANGE_VIRTUAL_DISPLAY
SMG\$CHECK_FOR_OCCLUSION	SMG\$CONTROL_MODE
SMG\$COPY_VIRTUAL_DISPLAY	SMG\$CREATE_MENU
SMG\$CREATE_PASTEBOARD	SMG\$CREATE_SUBPROCESS
SMG\$CREATE_VIEWPORT	SMG\$CREATE_VIRTUAL_DISPLAY
SMG\$CURSOR_COLUMN	SMG\$CURSOR_ROW
SMG\$DELETE_CHARS	SMG\$DELETE_LINE
SMG\$DELETE_MENU	SMG\$DELETE_PASTEBOARD
SMG\$DELETE_SUBPROCESS	SMG\$DELETE_VIEWPORT
SMG\$DELETE_VIRTUAL_DISPLAY	SMG\$DISABLE_BROADCAST_TRAPPING
SMG\$DISABLE_UNSOLICITED_INPUT	SMG\$DRAW_CHAR
SMG\$DRAW_LINE	SMG\$DRAW_RECTANGLE
SMG\$ENABLE_UNSOLICITED_INPUT	SMG\$END_DISPLAY_UPDATE
SMG\$END_PASTEBOARD_UPDATE	SMG\$ERASE_CHARS
SMG\$ERASE_COLUMN	SMG\$ERASE_DISPLAY
SMG\$ERASE_LINE	SMG\$ERASE_PASTEBOARD
SMG\$EXECUTE_COMMAND	SMG\$FIND_CURSOR_DISPLAY
SMG\$FLUSH_BUFFER	SMG\$GET_BROADCAST_MESSAGE
SMG\$GET_DISPLAY_ATTR	SMG\$GET_PASTEBOARD_ATTRIBUTES
SMG\$GET_PASTING_INFO	SMG\$GET_VIEWPORT_CHAR
SMG\$HOME_CURSOR	SMG\$INSERT_CHARS
SMG\$INSERT_LINE	SMG\$INVALIDATE_DISPLAY
SMG\$LABEL_BORDER	SMG\$LIST_PASTEBOARD_ORDER
SMG\$LIST_PASTING_ORDER	SMG\$LOAD_VIRTUAL_DISPLAY

(continued on next page)

Overview of the Chinese Screen Management Facility (SMG\$)

Table 1–3 (Cont.) Output Routines

Output Routines	
SMG\$MOVE_TEXT	SMG\$MOVE_VIRTUAL_DISPLAY
SMG\$PASTE_VIRTUAL_DISPLAY	SMG\$POP_VIRTUAL_DISPLAY
SMG\$PUT_CHARS	SMG\$PUT_CHARS_HIGHWIDE
SMG\$PUT_CHARS_MULTI	SMG\$PUT_CHARS_WIDE
SMG\$PUT_HELP_TEXT	SMG\$PUT_LINE
SMG\$PUT_LINE_HIGHWIDE	SMG\$PUT_LINE_MULTI
SMG\$PUT_LINE_WIDE	SMG\$PUT_STATUS_LINE
SMG\$READ_FROM_DISPLAY	SMG\$REMOVE_LINE
SMG\$REPAINT_SCREEN	SMG\$REPASTE_VIRTUAL_DISPLAY
SMG\$RESTORE_PHYSICAL_SCREEN	SMG\$RETURN_CURSOR_POS
SMG\$RING_BELL	SMG\$SAVE_PHYSICAL_SCREEN
SMG\$SAVE_VIRTUAL_DISPLAY	SMG\$SCROLL_DISPLAY_AREA
SMG\$SCROLL_VIEWPORT	SMG\$SELECT_FROM_MENU
SMG\$SET_BROADCAST_TRAPPING	SMG\$SET_CURSOR_ABS
SMG\$SET_CURSOR_MODE	SMG\$SET_CURSOR_REL
SMG\$SET_DISPLAY_SCROLL_REGION	SMG\$SET_OUT_OF_BAND_ASTS
SMG\$SET_PHYSICAL_CURSOR	SMG\$SET_TERM_CHARACTERISTICS
SMG\$UNPASTE_VIRTUAL_DISPLAY	

Note

The Chinese Screen Management Facility supports 4-byte input and output in the suppress leading code mode only. The display terminal must be set in the suppress leading code mode to display Chinese characters correctly. Chinese SMG does not automatically set up the display terminal to the suppress leading code mode. Users need to set up the display terminal in the suppress leading code mode by themselves beforehand.

As a result of the 4-byte suppress leading code support, the suppress leading code length of a text may be shorter or equal to its physical length.

The Chinese Screen Management Facility provides two important services.

- **Terminal Independence**

The Chinese screen management routines provide terminal independence by allowing you to perform commonly needed screen functions without concern for the type of terminal being used. All operations, including input and output, are performed by calling a routine that converts the caller's terminal-independent request (for example, to scroll a part of the screen) into the sequence of codes needed to perform that action. If the terminal being used does not support the requested operation in hardware, in most cases the Chinese screen management routines accomplish the action by emulating it

Overview of the Chinese Screen Management Facility (SMG\$)

in software. Similarly, the Chinese screen management routines provide a terminal-independent means for performing input from a keyboard without concern for the type of keyboard being used.

Note

The Chinese Screen Management Facility assumes that it has complete control of the terminal. Applications should not mix calls to SMG\$ with calls to other screen products such as DEC GKS or VAX FMS.

- Ease of Composition

The Chinese screen management routines assist you in composing complex images on a screen. For example, you may want to solicit user input from one part of the screen, display results on a second part of the screen, and maintain a status display in a third part of the screen. Normally, each routine that reads from or writes to one of these regions must be aware that other regions exist and know where on the screen they are positioned, in order to properly bias its row and column references to locate the display on the desired part of the screen. Using the Chinese screen management routines, a routine can independently write to its dedicated region of the screen without regard to the position of the region. References to row and column pertain only to the region of the screen the routine is addressing.

The following sections discuss the fundamental elements of screen management. These elements are the pasteboard, the virtual display, the viewport, and the virtual keyboard.

1.1 Pasteboards

A pasteboard is a logical structure for performing output operations to a terminal screen. You can think of a pasteboard as a two-dimensional area on which you place and manipulate screen displays. A pasteboard is always associated with a physical device, but a pasteboard may be larger or smaller than the physical screen. There can be only one pasteboard for each output device.

Note

The Chinese Screen Management Facility does not support an RMS file and a printer for the output device.

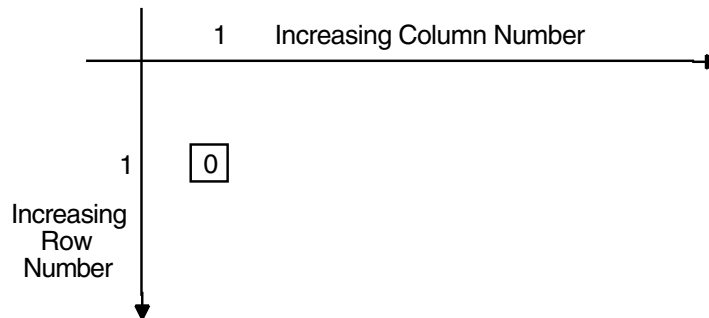
You create a pasteboard by calling the SMG\$CREATE_PASTEBOARD routine and specifying as an argument the physical device to be associated with the pasteboard. SMG\$CREATE_PASTEBOARD returns a unique pasteboard identifier (**pasteboard-id**), which is used in subsequent routine calls where a pasteboard identifier is needed. For example, you use the **pasteboard-id** to specify the physical terminal screen on which to paste a virtual display. SMG\$CREATE_PASTEBOARD also returns as output arguments the numbers of rows and columns available on the associated device. You can use this information to create a virtual display of the size of the physical screen. (Virtual displays are discussed in the next section.)

Overview of the Chinese Screen Management Facility (SMG\$)

1.1 Pasteboards

It is useful to think of a pasteboard as a logical coordinate system in which the relative orientation of one or more virtual displays is specified. (The pasteboard itself has no physical boundaries, but the physical screen does.) Figure 1-1 depicts the pasteboard coordinate system.

Figure 1-1 Pasteboard Coordinate System



ZK-1910-GE

The origin (cellular position 1,1) corresponds to the upper left-hand corner of the physical screen. The numbering of rows and columns starts from this origin. For example, on a VT200 series terminal, with 24 rows and 80 columns, the first 24 rows and first 80 columns of the pasteboard coordinate system map to the physical screen. Note that you can place a virtual display anywhere in this coordinate system, not only in the quadrant that corresponds to the physical screen. Thus a virtual display, when pasted (that is, positioned on the pasteboard), may be invisible or only partly visible on the physical screen.

Pasteboards are deleted, or disassociated, from a particular device by the `SMG$DELETE_PASTEBOARD` routine. When a pasteboard is deleted, all virtual displays pasted to it are unpasted.

Once a pasteboard has been created, you can learn about its attributes (particularly its dimensions) by calling `SMG$GET_PASTEBOARD_ATTRIBUTES`. You can change the characteristics of a pasteboard by calling `SMG$CHANGE_PBD_CHARACTERISTICS` if the associated physical device allows the change. For example, if the device is a VT100, you can change the width of the pasteboard from 80 columns to 132 columns.

When the pasteboard is created, the Chinese Screen Management Facility clears the screen by default; however, you can request that the screen be left as it is. In addition, you can call `SMG$ERASE_PASTEBOARD` to erase the screen explicitly.

1.2 Virtual Displays

A virtual display is a rectangular part of the terminal screen to which a program writes data using routine calls. Virtual displays are the main focus of the Chinese Screen Management Facility. When you create images to be placed on the screen, you should think in terms of virtual displays rather than in terms of the physical screen. This logical separation of the virtual display from the physical screen allows a main program to reposition virtual displays, so that a subroutine that

Overview of the Chinese Screen Management Facility (SMG\$)

1.2 Virtual Displays

writes to the virtual display need not be involved with positioning the display on the physical screen.

When a virtual display is associated with a pasteboard, it is said to be **pasted**. When the display is removed from the pasteboard, it is said to be **unpasted**. A virtual display is not displayed unless it is pasted to a pasteboard. (See Section 2.1.1 for more information on pasting virtual displays.)

A program can create and maintain any number of virtual displays (limited only by the virtual address space available). A single virtual display can be pasted to more than one pasteboard at a time; thus, a program only needs to maintain the virtual display. Any change to a virtual display is automatically reflected in each pasteboard to which the display is pasted (and its associated terminal screen).

You create a virtual display by calling the `SMG$CREATE_VIRTUAL_DISPLAY` routine. A call to this routine must specify the number of rows and columns that make up the virtual display. The program can also request certain display and video attributes to be applied to the display.

`SMG$CREATE_VIRTUAL_DISPLAY` returns a unique virtual display identifier (**display-id**). This **display-id** is used to identify the virtual display in subsequent routine calls that modify the display.

A program or subroutine can determine which attributes and dimensions are associated with a virtual display by calling the `SMG$GET_DISPLAY_ATTR` routine. If you have multiple virtual displays pasted to a pasteboard, you can use `SMG$LIST_PASTING_ORDER` to determine the order in which the virtual displays are pasted.

The default video characteristics are the characteristics applied to the output when no other attributes have been specified. **Renditions** are video characteristics that you can turn on or off; they include bolding, blinking, reverse video, and underlined text. Display attributes are the characteristics that specify whether or not the display

- Is bordered (the border may be labeled)
- Echoes carriage control characters (like form feed, vertical tab, and so on).
- Shows the user a diamond-shaped icon when text extends past the rightmost position in the display

The video and display attributes you specify when you create a virtual display can be changed. The `SMG$CHANGE_RENDITION` routine lets you change video attributes while the `SMG$CHANGE_VIRTUAL_DISPLAY` routine lets you change both video and display attributes. For example, you can redimension a virtual display with the latter routine. When you redimension a virtual display, the data in it is copied to the redimensioned display; that is, as much of the current contents (starting with row 1, column 1) as will fit in the newly dimensioned display are preserved.

You can delete a virtual display by calling the `SMG$DELETE_VIRTUAL_DISPLAY` routine. See Section 2.1.5 for more information on the delete operation.

1.3 Viewports

Since a virtual display can be very large, it is not always possible to show the entire display on the screen at one time. You must repaste a large virtual display in order to view a different portion of it. A viewport associated with the virtual display makes this job easier.

Viewporting refers to the process of moving a rectangular viewing area around on a virtual display in order to view different pieces of the virtual display. The viewport is associated with the virtual display so that any output operation that is performed on the virtual display is reflected on the viewport. You can create, delete, paste, unpaste, scroll, and move a viewport. See Section 2.2.10 for more information on viewports.

1.4 Virtual Keyboards

A virtual keyboard is a logical structure for input operations, just as a pasteboard is a logical structure for output operations. The advantage of using virtual keyboards is device independence. When using the screen management input routines, you need not worry about the type of terminal being used. For example, your program need not know which line terminators a particular terminal uses; the Chinese screen management routines map the different terminator character sequences into a uniform set of function codes. (See Chapter 3 for more information about terminator codes.)

A virtual keyboard is usually associated with a physical keyboard on a terminal. There is a many-to-one correspondence between virtual keyboards and an input device.

Note

The Chinese Screen Management Facility does not support an RMS file for the input device.

You establish a source for input (a virtual keyboard) by calling the `SMG$CREATE_VIRTUAL_KEYBOARD` routine. You delete virtual keyboards by calling the `SMG$DELETE_VIRTUAL_KEYBOARD` routine. Once you have created a virtual keyboard, you can obtain data from it with the `SMG$READ_COMPOSED_LINE`, `SMG$READ_KEYSTROKE`, or `SMG$READ_STRING` routine. `SMG$READ_COMPOSED_LINE` reads a line composed of ordinary keystrokes and predefined strings associated with keypad and control keys; it provides an easy way to code an interface for command-oriented utilities by providing single-key command capabilities. `SMG$READ_KEYSTROKE` is used to read one keystroke entered at the keyboard. `SMG$READ_STRING` reads a string composed of characters and a terminator; this routine is for general purpose and flexible, providing access to many features of the OpenVMS terminal driver.

Chinese Screen Management Output Operations

This chapter discusses the output operations provided by the Screen Management Facility. These output operations are described in terms of composition operations (operations that, in effect, create an image on a terminal screen) and output operations through virtual displays.

2.1 Composition Operations

Composition operations are the routines you use to manipulate virtual displays on a pasteboard and thus to create an image on a terminal screen. These operations include pasting, unpasting, repasting, moving and popping virtual displays, checking virtual displays for occlusion, and listing the pasting order.

2.1.1 Paste Operation

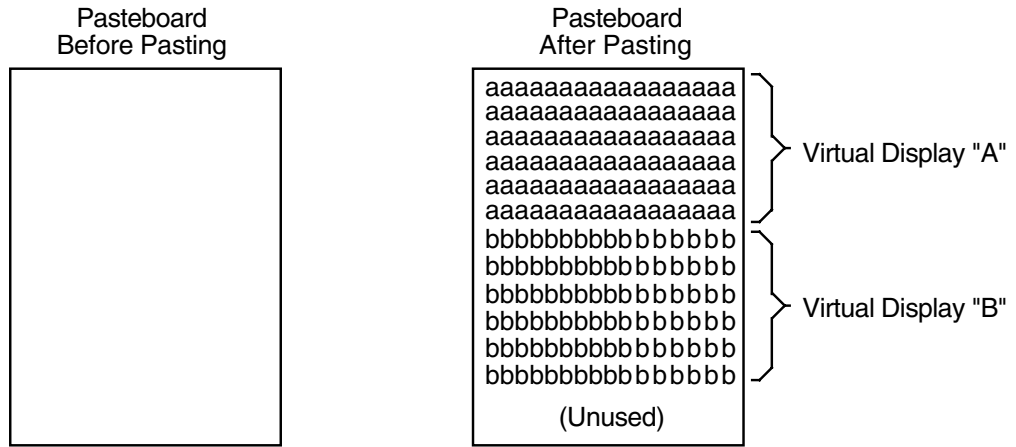
Virtual displays are visible on a physical device only while they are pasted to a pasteboard. You paste a display to a pasteboard by calling `SMG$PASTE_VIRTUAL_DISPLAY` and specifying the pasteboard coordinates to be used as the origin of the virtual display. (The origin is the top left-hand corner.) The pasteboard itself has no boundaries, but of course the physical screen does. Thus you can paste a display to a pasteboard in such a way that some or all of the display does not appear on the terminal screen.

Pasting virtual displays to a pasteboard is a logical operation that maps the contents of a virtual display to a location on the screen by specifying the row and column of the pasteboard that coincide with row 1 and column 1 of the virtual display. For example, pasting a 6-row virtual display “A” to pasteboard rows 1 through 6 and pasting a second 6-row virtual display “B” to pasteboard rows 7 through 12 places virtual display “B” immediately below virtual display “A” on the screen. See Figure 2–1.

Chinese Screen Management Output Operations

2.1 Composition Operations

Figure 2-1 Paste Operation

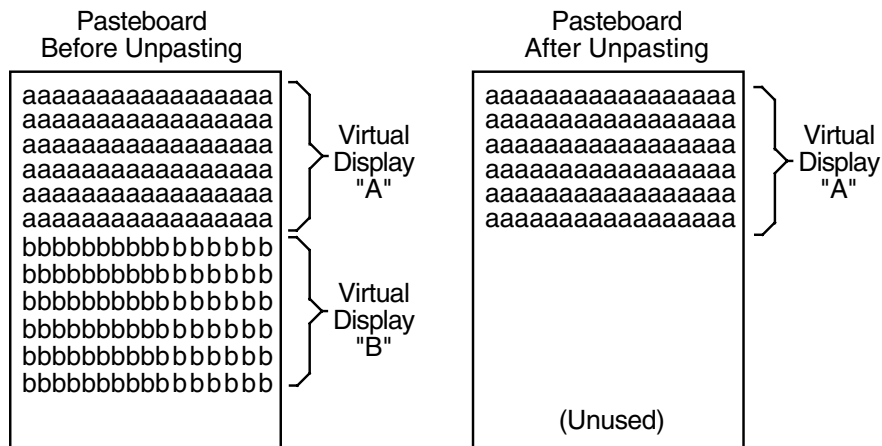


ZK-1911-GE

2.1.2 Unpaste Operation

A virtual display can be made to disappear from the physical screen with the `SMG$UNPASTE_VIRTUAL_DISPLAY` routine. To continue the example in Section 2.1.1, if virtual display "B" is unpasted, the results appear as in Figure 2-2.

Figure 2-2 Unpaste Operation



ZK-1912-GE

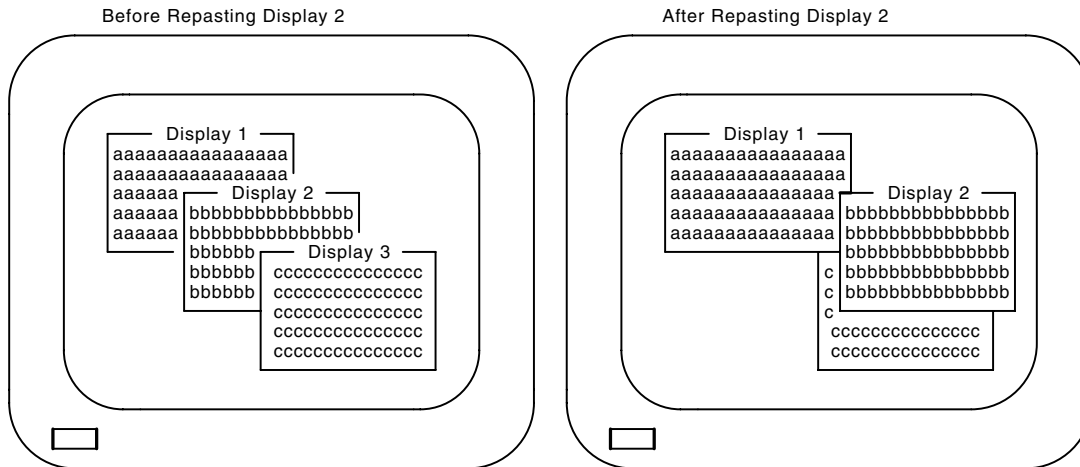
Unpasting a virtual display does not destroy the virtual display or its contents; it simply removes the display from the pasteboard.

Displays can overlap partially or completely, depending on their size, where they are pasted, and the order in which they are pasted. This overlap is called **occlusion**. Unpasting the top display causes the underlying display(s) to be visible.

2.1.3 Repaste Operation

You can move a virtual display to a new location on the pasteboard by calling `SMG$REPASTE_VIRTUAL_DISPLAY`, which prevents the screen from being left blank during the unpaste and repaste operations. Figure 2–3 below shows the effect of repasting the second display farther to the right. Notice that display 2 has been pulled out of its former pasting order and is now uppermost — hiding part of display 3, which was uppermost before the repasting operation.

Figure 2–3 Repaste Operation



ZK-1918-GE

2.1.4 Move Operation

You can also move a virtual display around the pasteboard while preserving its pasting order by calling the `SMG$MOVE_VIRTUAL_DISPLAY` routine. Figure 2–4 shows the effect of moving the second display to the right. Note the difference between the unpaste and move operations: the pasting order does not change with a move. Thus, display 2 remains partially occluded by display 3.

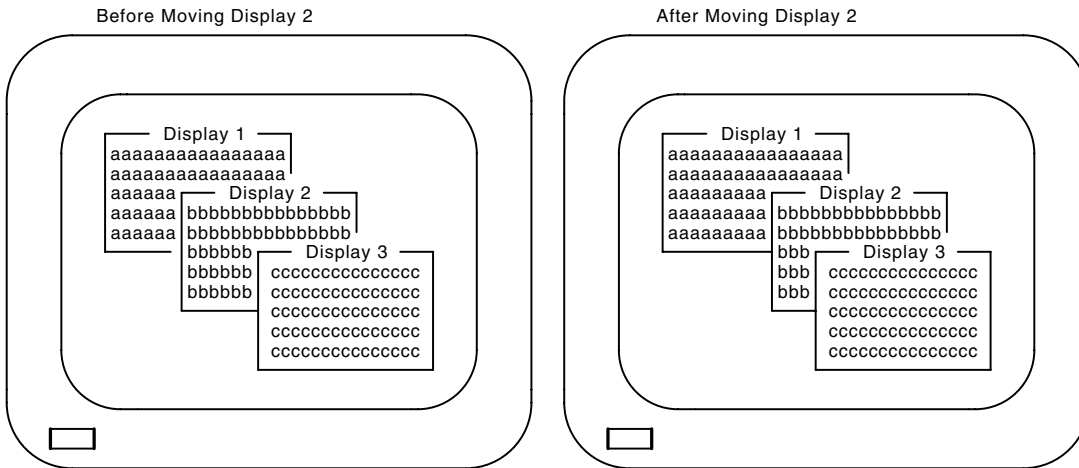
The routine `SMG$MOVE_TEXT` allows you to move text from one virtual display to another virtual display. Given two points in opposite corners of the rectangle, `SMG$MOVE_TEXT` determines the desired width and height. The attributes of the first virtual display are moved, and after the rectangle of text is moved, it is erased from the first virtual display.

In case that this operation divides characters, the divided characters are treated as undefined characters.

Chinese Screen Management Output Operations

2.1 Composition Operations

Figure 2-4 Move Operation



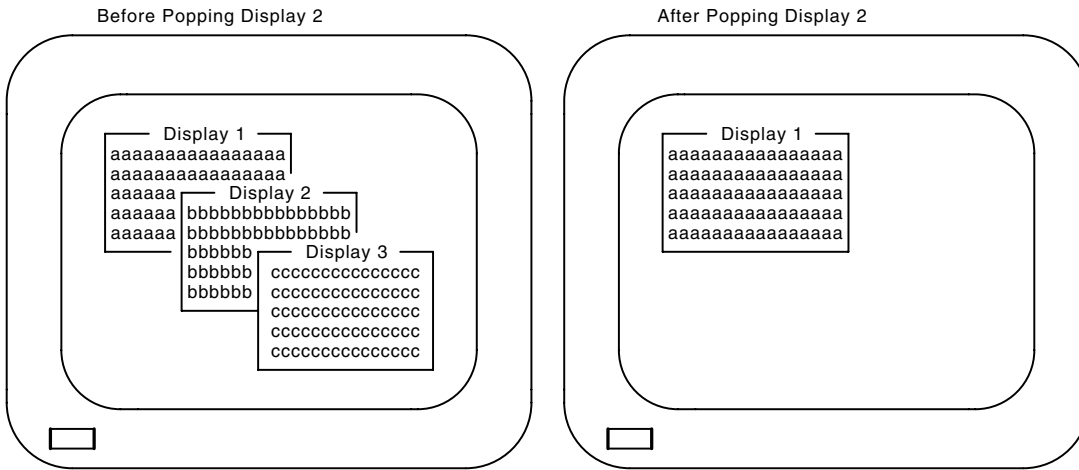
ZK-1913-GE

2.1.5 Delete and Pop Operations

The unpaste, repaste, and move operations shown thus far do not destroy the virtual displays affected. You can remove and delete a virtual display by calling the `SMG$DELETE_VIRTUAL_DISPLAY` routine. You can also remove a number of virtual displays from a pasteboard and delete them in a single operation by calling `SMG$POP_VIRTUAL_DISPLAY`. This routine unpastes and deletes the specified virtual display and all other virtual displays that were pasted after the one specified.

The pop operation is useful in a modular environment. For example, you can call a subroutine and pass only the **pasteboard-id** upon which it is to produce output. The subroutine can then create additional virtual displays and paste them to the indicated pasteboard. When the subroutine returns control to its caller, the subroutine returns the **display-id** of the first virtual display it has pasted. The calling program can then undo the effects of the subroutine by calling `SMG$POP_VIRTUAL_DISPLAY`, passing the identifier of the virtual display returned by the subroutine. This technique minimizes the amount of information that needs to be passed between the calling program and its subroutine. Figure 2-5 shows the effects of popping display 2.

Figure 2-5 Pop Operation



ZK-1914-GE

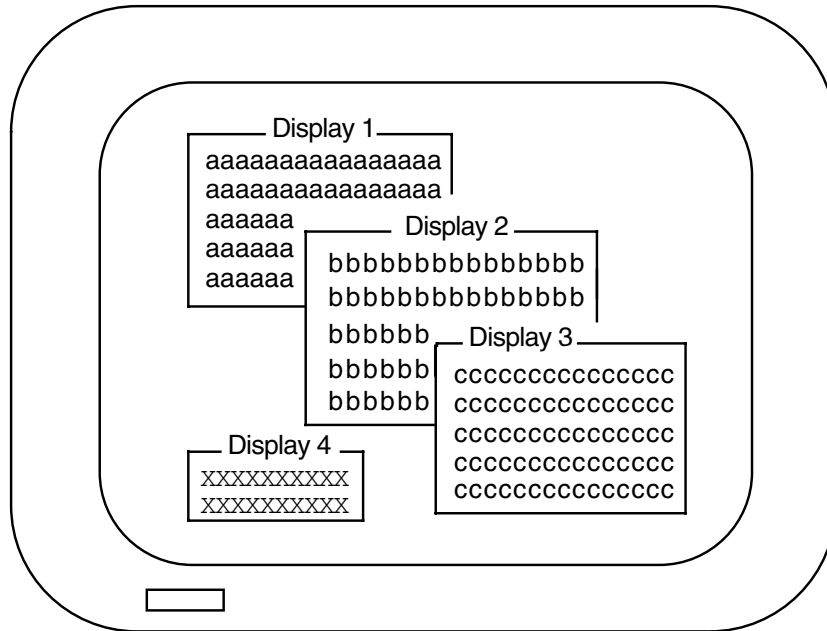
2.1.6 Occlusion Check Operation

It is sometimes useful to determine whether a display is occluded, as pasted on a given pasteboard. You can find this out by calling the `SMG$CHECK_FOR_OCCLUSION` routine. For example, in the configuration represented in Figure 2-6, displays 1 and 2 would be reported as being occluded, while displays 3 and 4 would be reported as not occluded. Note that this test cannot be used to determine which display is pasted uppermost on the pasteboard; it can determine only whether or not the display, as pasted, is occluded.

Chinese Screen Management Output Operations

2.1 Composition Operations

Figure 2-6 Occlusion Check



ZK-1915-GE

If you have multiple virtual displays pasted to a pasteboard, you can use `SMG$LIST_PASTING_ORDER` to determine the order in which virtual displays are pasted. This routine returns the identifier of the first, or bottommost, virtual display pasted. You call `SMG$LIST_PASTING_ORDER` in a loop until the identifiers of all the succeeding pasted virtual displays are returned.

The routine `SMG$LIST_PASTEBOARD_ORDER` gives you the inverse of the information returned by `SMG$LIST_PASTING_ORDER`. `SMG$LIST_PASTEBOARD_ORDER` returns the identifier of the first, or bottommost, pasteboard to which the specified virtual display is pasted.

2.2 Output Through Virtual Displays

This section describes the Chinese screen management routines used to perform output through virtual displays.

Writing to a virtual display is similar to writing directly to the terminal. However, writing to a virtual display is done entirely by calling screen management routines. Erasing the screen, setting the cursor position, and scrolling output text are typical operations provided by the Chinese Screen Management Facility. Text is arranged in the virtual display's buffer, so the display need not be pasted before it can receive output. When you write to the physical screen, you are limited by the physical boundaries of the screen. Similarly, Chinese screen management output operations are confined to the boundaries of the virtual display: you cannot write text beyond the last column of a virtual display.

Remember that changes to a virtual display are not seen on the screen unless the virtual display is pasted to the part of the pasteboard that is visible on the screen. If the virtual display is not pasted, or if it is pasted in a position that is not visible, such changes are reflected only in the internal database that represents the virtual display.

2.2.1 Cursor Position

When a virtual display is first created, the virtual cursor is positioned at row 1, column 1 of the virtual display. Various output operations to the virtual display move the virtual cursor, just as output operations do on a physical terminal.

The position of the virtual cursor in a virtual display should not be confused with the position of the physical cursor on the screen. There may be many virtual displays pasted to a pasteboard and hence visible at the same time on the physical screen. Although each virtual display has an associated virtual cursor position, only one of the virtual cursor positions for all these displays corresponds to the physical cursor—usually the cursor position of the virtual display that has been modified most recently.

You can determine the current position of the virtual cursor within a virtual display by calling the `SMG$RETURN_CURSOR_POS` routine. This routine returns the current virtual cursor row and column.

For programming convenience, this information can also be obtained through two separate routines, `SMG$CURSOR_ROW` and `SMG$CURSOR_COLUMN`, which operate as functions. These two routines make it easy to code constructions like this:

```
IF SMG$CURSOR_ROW ( Display-id ) > Max-row
THEN
  BEGIN
    .
    .
    .
  END
```

To obtain this information with `SMG$RETURN_CURSOR_POS`, you would write the following:

```
CALL SMG$RETURN_CURSOR_POS ( Display-id, Cursor-row, Cursor-column )
IF Cursor-row > Max-row
THEN
  BEGIN
    .
    .
    .
  END
```

`SMG$RETURN_CURSOR_POS` requires you to define two local variables, *cursor-row* and *cursor-column*, which you might not need except to perform this test. However, this routine yields both the row and column in a single routine call.

The three following routines are available to set the virtual cursor position in a virtual display:

- The `SMG$SET_CURSOR_ABS` routine sets the virtual cursor to the specified position in the virtual display.
- The `SMG$SET_CURSOR_REL` routine sets the virtual cursor position to the specified offset from the current display cursor position.
- The `SMG$HOME_CURSOR` routine sets the virtual cursor to the virtual display's home position (row 1, column 1).

Chinese Screen Management Output Operations

2.2 Output Through Virtual Displays

2.2.2 Deletion Operations

Two routines are provided to delete parts of a virtual display.

SMG\$DELETE_CHARS deletes one or more characters on a single line. Character positions removed by this routine are replaced with the characters to the right of the deleted characters on the same line. Character positions opened at the end of the line are filled with blanks.

In case of using Chinese characters, if delete operation starts from the right portion of the character, the left portion is treated as an undefined character; and if the delete operation ends at the left portion of the character, the right portion is treated as an undefined character.

SMG\$DELETE_LINE deletes one or more entire lines. Lines removed by this routine are filled by the lines immediately below the deleted lines. New lines introduced into the bottom of the virtual display are blank.

2.2.3 Erasure Operations

During an erase operation, the erased portion of the virtual display is filled with blanks. No other parts of the virtual display are rearranged.

In case of Chinese characters, this operation may start from the right portion of a character or end at the left portion. If it starts from the right portion of the character, the left portion is treated as an undefined character. If it terminates on the left portion of the character, the right portion is treated as an undefined character.

Four routines are provided to erase parts of a virtual display:

- SMG\$ERASE_CHARS erases a specified number of columns within a given line.
- SMG\$ERASE_COLUMN erases the specified portion of the virtual display from the given position to the end of the column.
- SMG\$ERASE_LINE erases characters in a line from the specified starting position to the end of the line.
- SMG\$ERASE_DISPLAY erases all or part of a virtual display.

2.2.4 Insertion Operations

Two routines are provided to insert text into a virtual display.

In case of Chinese characters, this operation may divide a character. In such case, those divided portions of the character are treated as undefined characters.

SMG\$INSERT_CHARS deposits the specified string of characters in the indicated starting position. Existing characters in these positions are shifted to the right to make room for each character as it is inserted. Characters shifted beyond the rightmost column are discarded.

SMG\$INSERT_LINE inserts the specified line of text in the position indicated and scrolls existing lines in the virtual display up or down to make room for the inserted lines. Lines scrolled above the top line or below the bottom line of the virtual display are discarded.

2.2.5 Writing Operations

The Chinese Screen Management Facility provides two types of routines for writing text to a virtual display: character-oriented output and line-oriented output. The following sections describe these routines.

2.2.5.1 Character-Oriented Output

You typically use the character-oriented output routines when using a virtual display as a direct-access device. In this mode of operation, the program explicitly sets the cursor in the virtual display and deposits text there. Since the next output operation usually has no spatial relationship to the previous one, you need to control the cursor position and display scrolling explicitly.

This operation may overwrite a portion of a Chinese character. In this case, the portion left is treated as an undefined character.

There are four character-oriented output routines:

- `SMG$PUT_CHARS` writes normal characters to a virtual display.
- `SMG$PUT_CHARS_WIDE` writes double-width characters to a virtual display.
- `SMG$PUT_CHARS_HIGHWIDE` writes double-width, double-height characters to a virtual display.
- `SMG$PUT_CHARS_MULTI` writes characters with multiple renditions to the virtual display.

Note that you cannot mix different types of characters on a single line in a virtual display.

2.2.5.2 Line-Oriented Output

In contrast to the character-oriented output routines, the line-oriented routines treat a terminal as a sequential device. In this mode of operation, the program typically writes one line of information after another. Conceptually, this action corresponds to copying a stream of information (for example, a file) to a virtual display. Each routine call leaves the cursor at column 1 of the next row after the operation is complete.

This operation may start in the middle of a Chinese character. In such case, the left portion of the character is treated as an undefined character.

There are four line-oriented output routines:

- `SMG$PUT_LINE` writes lines of text to a virtual display.
- `SMG$PUT_LINE_WIDE` writes lines of double-width text to a virtual display.
- `SMG$PUT_LINE_HIGHWIDE` writes lines of double-width, double-height text to a virtual display.
- `SMG$PUT_LINE_MULTI` writes lines with multiple renditions to the virtual display.

Chinese Screen Management Output Operations

2.2 Output Through Virtual Displays

2.2.6 Changing the Rendition of a Virtual Display

When you create a virtual display with the `SMG$CREATE_VIRTUAL_DISPLAY` routine, you specify a default rendition for all text that appears in the virtual display. You can change the rendition for an existing virtual display by calling either the `SMG$CHANGE_VIRTUAL_DISPLAY` or `SMG$CHANGE_RENDITION` routines.

The `SMG$CHANGE_VIRTUAL_DISPLAY` routine lets you change display attributes as well as video attributes for the entire display; the `SMG$CHANGE_RENDITION` routine can be used to change the video rendition of text already in the virtual display. For example, a program may maintain on the screen a list of values that change cyclically. When a number first changes, it can be displayed in reverse video to highlight it as a change on that cycle. On the next cycle, the same number must be displayed, but the reverse video should be removed, since the value of the number did not change. `SMG$CHANGE_RENDITION` provides an easy way to perform such changes.

Another use for the `SMG$CHANGE_RENDITION` routine is in implementing menus. Menu choices can be painted on the screen and the current choice highlighted by some video attribute, such as blinking characters or reverse video. As the user moves a cursor to change the selection, the rendition of a menu item can be altered so that the current selection is always highlighted. Such changes in rendition can be made independently of the text contained in the menu choices.

To specify the default rendition for a virtual display, you use bit masks to set bits in the display attributes argument. The following bits can be set:

<code>SMG\$M_BLINK</code>	Specifies blinking characters.
<code>SMG\$M_BOLD</code>	Specifies characters in higher-than-normal intensity.
<code>SMG\$M_REVERSE</code>	Specifies characters in reverse video; that is, the opposite of the current rendition of the virtual display.
<code>SMG\$M_UNDERLINE</code>	Specifies underlined characters.
<code>SMG\$M_INVISIBLE</code>	Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard.
<code>SMG\$M_USER1</code> through <code>SMG\$M_USER8</code>	Specifies a user-defined rendition.

In order to use one of the user-defined renditions `SMG$M_USER1` through `SMG$M_USER8`, you must provide an appropriate definition in the file `TERMTABLE.TXT`, using `STRING_2` capabilities. The `TERMTABLE` definitions and `STRING_2` capabilities are discussed in Chapter 5.

Any or all of the characteristics listed previously can be specified in the rendition of a virtual display. To specify more than one video attribute, you use the logical OR of these characteristics. For example, to specify underlined characters in reverse video as the default for a virtual display, you assign the logical OR of the appropriate bit masks to the **display-attributes** argument:

```
Display_attributes = ( SMG$M_REVERSE OR SMG$M_UNDERLINE )
```

You then pass this **display-attributes** argument in the call to the `SMG$CREATE_VIRTUAL_DISPLAY` routine.

Chinese Screen Management Output Operations

2.2 Output Through Virtual Displays

Chinese Screen management output routines let you override the default rendition so that you need not change the default each time you want to write text in some other rendition. Two arguments provide the means to override the default rendition: **rendition-set** and **rendition-complement**. The scheme for setting video attributes in these arguments is the same as that for setting the video attributes when you are creating a virtual display.

The default video attributes, the **rendition-set** argument, and the **rendition-complement** argument together specify the output rendition according to the following scheme:

1. The logical or bitwise OR operation is performed on the mask containing the default video attributes and the **rendition-set** argument.
2. The logical or bitwise EXCLUSIVE OR operation is performed on the result of the previous OR operation and the **rendition-complement** argument.

The results of this scheme are shown in the following table.

Set	Complement	Action
0	0	Attribute set to default
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

Note that the effect of this scheme depends on the default attribute setting, not the current rendition of the virtual display. Thus, if you have used screen management output routines that explicitly specify a rendition, the current rendition may not match the default rendition for that virtual display.

2.2.7 Drawing and Removing Drawn Lines and Characters

Three routines provide a simple way to construct horizontal and vertical lines. `SMG$DRAW_LINE` constructs either horizontal or vertical lines, given the end points of those lines. `SMG$DRAW_RECTANGLE` draws a rectangle given the position of the upper left-hand corner and the lower right-hand corner. `SMG$DRAW_CHAR` draws one line-drawing character.

If you want to erase a line drawn with `SMG$DRAW_LINE` or `SMG$DRAW_RECTANGLE`, use `SMG$REMOVE_LINE`. This routine removes the line but preserves the line-drawing characters at any line intersections.

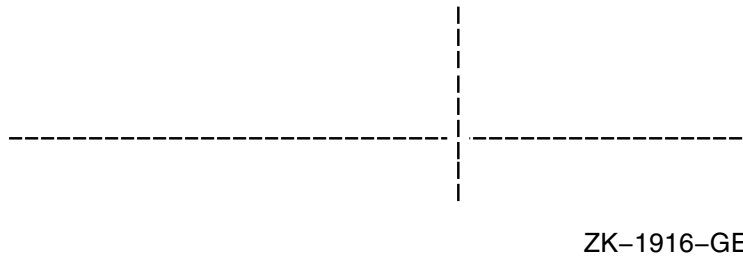
Like all Chinese screen management routines, these are device independent. If the resulting line is to be drawn on a VT100 terminal, the VT100 line-drawing character set is used. If the same line is drawn on a VT52 (which does not have this hardware capability) terminal, the lines will automatically be approximated by the use of the plus sign (+), the vertical bar (|), and the dash (-). Your program does not have to supply different character codes for different types of terminals.

In addition, these routines automatically provide an appropriate character at the intersection of two lines. For example, if a program writes a horizontal line directly to the screen and then writes a vertical line that intersects the horizontal line, you would normally see what appears in Figure 2-7:

Chinese Screen Management Output Operations

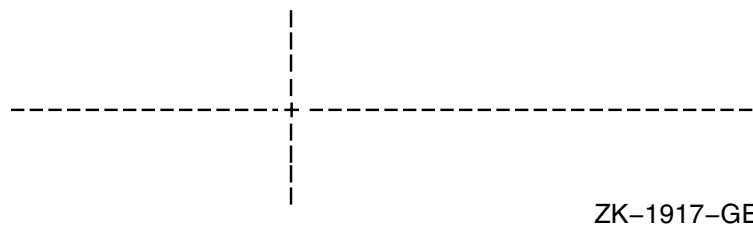
2.2 Output Through Virtual Displays

Figure 2-7 Lines Drawn Without SMG\$DRAW_LINE



If these same lines are drawn using SMG\$DRAW_LINE, the screen shows what appears in Figure 2-8:

Figure 2-8 Lines Drawn with SMG\$DRAW_LINE



2.2.8 Displaying External Text

Two routines provide a way to output “external” text to the virtual display or terminal.

The routine SMG\$PUT_HELP_TEXT outputs the help text for the specified topic in the virtual display provided.

The routine SMG\$PUT_STATUS_LINE outputs a line of text to the terminal’s hardware status line. Some terminals have a hardware status line at the bottom (25th line) of the screen. If this line has been set as “host writable”, you can use this routine to output a line of text in reverse video to the status line.

2.2.9 Reading from a Virtual Display

The SMG\$READ_FROM_DISPLAY routine is provided to make it easy to obtain text from a virtual display. This routine might be used in applications that present menu items on the screen by way of a virtual display. The application might allow the user to move the cursor among the menu items and then select one (by pressing the Return key, for example). At this point, the program can read characters from the display at the current cursor position and determine which menu item was selected. Note that this routine also provides a way to read characters written with the SMG\$M_INVISIBLE attribute.

2.2.10 Viewports

Since a virtual display can be very large, it is not always possible to show the entire display on the screen at one time. The user must repaste a large virtual display in order to view a different portion of it. A viewport associated with the virtual display makes this job easier.

Chinese Screen Management Output Operations

2.2 Output Through Virtual Displays

Viewporting refers to the process of moving a rectangular viewing area around on a virtual display in order to view different pieces of the virtual display. The viewport is associated with the virtual display so that any output operation performed on the virtual display is reflected on the viewport.

2.2.10.1 Creating a Viewport

The SMG\$CREATE_VIEWPORT routine creates a viewport associated with a particular virtual display. The virtual display must be created before the viewport can be created, and you can only create one viewport for each virtual display. In order to make the viewport visible, you have to paste the virtual display by calling the SMG\$PASTE_VIRTUAL_DISPLAY routine; only the portion of the virtual display that falls inside the viewport is visible.

2.2.10.2 Deleting a Viewport

You delete a viewport with the SMG\$DELETE_VIEWPORT routine. When you invoke this routine, the viewport is automatically unpasted from any pasteboards to which it is pasted. It is important to note, however, that the virtual display associated with the viewport has not been deleted. You can make the virtual display visible by calling SMG\$PASTE_VIRTUAL_DISPLAY.

2.2.10.3 Pasting and Unpasting a Viewport

The routine SMG\$PASTE_VIRTUAL_DISPLAY pastes either a viewport or a virtual display to a pasteboard. Once you have associated a viewport with a virtual display, any call to SMG\$PASTE_VIRTUAL_DISPLAY uses the viewport instead of the virtual display. That is, once a viewport for a virtual display is created, the only part of that virtual display that you can view is the rectangular region contained in the viewport. To unpaste a viewport without deleting it, you can invoke SMG\$UNPASTE_VIRTUAL_DISPLAY.

If you create a viewport when the associated virtual display is already pasted, the viewport will not be visible. A call to SMG\$PASTE_VIRTUAL_DISPLAY will unpaste the virtual display and paste the viewport in its place.

2.2.10.4 Scrolling and Moving a Viewport

A viewport associated with a virtual display may be situated entirely or partially on the pasteboard, or totally off the pasteboard. However, a viewport cannot extend beyond its associated virtual display. If you try to extend a viewport beyond the boundaries of its virtual display, the Chinese Screen Management Facility automatically truncates the viewport to fit into the virtual display.

In order to “scroll” a viewport, you scroll the virtual display associated with the viewport. You do this by calling SMG\$SCROLL_VIEWPORT. In actuality, the coordinates of the viewport are changing as it moves over the virtual display to simulate scrolling; however, the location of the viewport on the screen does not change. With the SMG\$SCROLL_VIEWPORT routine, you can specify the direction (up, down, left, or right) that you want to scroll.

You can move a viewport by calling SMG\$CHANGE_VIEWPORT. This routine lets you specify a new starting location and size for the viewport. By changing the starting location and size of the viewport, you can, in effect, move the window around the virtual display.

Chinese Screen Management Output Operations

2.2 Output Through Virtual Displays

2.2.10.5 Changing Viewport Characteristics

The routine `SMG$GET_VIEWPORT_CHAR` lets you retrieve the current characteristics of a viewport. The characteristics of a viewport consist of the starting and ending row and column positions for the viewport. You can use this routine in conjunction with the `SMG$CHANGE_VIEWPORT` routine, which lets you change the starting and ending positions of an existing viewport.

If you want to change any characteristic of a viewport other than its starting or ending position, you should use the `SMG$CHANGE_VIRTUAL_DISPLAY` routine. Any change you make to a virtual display will be reflected in its associated viewport.

For example, if a virtual display has a border, so does the associated viewport. If the virtual display does not have a border, then neither does the viewport. If you want to add or delete a border to a viewport, add or delete the border to the virtual display using the `SMG$CHANGE_VIRTUAL_DISPLAY` routine. This change is automatically reflected on the viewport.

2.2.11 Menus

The Chinese Screen Management Facility provides the capability to create and make selections from a menu. The menu can be a block menu, a vertical menu, or a horizontal menu. A block menu is a two-dimensional array of items and is the main type of menu provided. A vertical menu displays the menu choices in a single column, while a horizontal menu displays the choices in a single row. Any menu items that do not fit within the bounds of the viewport are not displayed until they are scrolled into view.

2.2.11.1 Creating a Menu

The routine `SMG$CREATE_MENU` creates a menu in the scrolling region of a specified virtual display. (By default, the scrolling region is the entire virtual display. You can use the routine `SMG$SET_DISPLAY_SCROLL_REGION` to change the scrolling region.) You specify a format for the menu (block, vertical, or horizontal) when you create it.

A block menu is the default format for a menu. The items in the menu are passed to the routine in the form of a static array of character strings. The menu choices are single spaced by default, but you can request double spacing. Four spaces separate each menu item horizontally. In addition, you can request that the menu choices be displayed in *fixed format* columns, where the width of the column is equal to the size of the fixed-length strings being passed.

It is important to note that each virtual display can only contain one menu. Also, after calling `SMG$CREATE_MENU`, you must not output any characters to the display that disturb the area containing the menu; otherwise, the results are unpredictable. The menu is output in the scrolling region of the virtual display.

2.2.11.2 Deleting a Menu

You delete a menu by a call to `SMG$DELETE_MENU`. This routine discontinues access to the menu choices in the specified virtual display. Additionally, you can request that `SMG$DELETE_MENU` remove all menu choices from the display when the menu is deleted.

Chinese Screen Management Output Operations

2.2 Output Through Virtual Displays

2.2.11.3 Selecting from a Menu

Once you have created a menu, you can select items from that menu using the `SMG$SELECT_FROM_MENU` routine. When you move around the menu items, the currently selected item is highlighted in reverse video by default. You can specify a default selection that is highlighted and becomes the current item when you call `SMG$SELECT_FROM_MENU`. If you do not specify a default selection item, the previously selected item remains highlighted.

`SMG$SELECT_FROM_MENU` provides three modes of operation; you can switch between these modes using the **flags** parameter. Each mode is described in the following sections.

2.2.11.3.1 Default Mode The default mode of operation for the `SMG$SELECT_FROM_MENU` routine is invoked by omitting the **flags** parameter. In this mode, you can move around the menu items using the arrow keys, and after selecting an item, you can continue making additional selections. The default mode also lets you “reselect” items that were already selected.

2.2.11.3.2 RETURN_IMMED Mode Specifying the `SMG%M_RETURN_IMMED` value for the **flags** parameter of the `SMG$SELECT_FROM_MENU` routine allows you to move around the menu choices with the arrow keys; however, pressing any other key returns control to the user. `Ctrl/Z` selects the current item and returns `SMG$_EOF`. Any other key entered selects the current item.

Use `SMG%M_RETURN_IMMED` mode if you want key definitions other than those provided by the default mode.

2.2.11.3.3 REMOVE_ITEM Mode If you specify the `SMG%M_REMOVE_ITEM` value for the **flags** parameter of `SMG$SELECT_FROM_MENU`, you cannot “reselect” an item in the menu, although the item remains in the menu. It appears in the default rendition for the virtual display containing the menu.

If you specify a default selection item while in this mode, and that item has already been selected, the first “selectable” item in the menu is highlighted. If none of the items is selectable, an error is returned.

2.2.12 Saving a Virtual Display

The routine `SMG$SAVE_VIRTUAL_DISPLAY` saves the contents of a virtual display in a file. The text, renditions, and all the attributes needed to reconstruct the virtual display are saved, but menu, viewport, and subprocess contexts are not saved. You cannot print the resulting file. To restore the virtual display, you can use `SMG$LOAD_VIRTUAL_DISPLAY`, which creates a new virtual display and loads it with the saved contents of the display. The new virtual display is not pasted to any pasteboard.

2.2.13 Changing Terminal Characteristics

The routine `SMG$SET_TERM_CHARACTERISTICS` changes or retrieves the terminal characteristics for a given pasteboard. With this routine, you can control multiple terminal characteristics in a single routine call.

Chinese Screen Management Output Operations

2.2 Output Through Virtual Displays

2.2.14 Setting the Default Character Set of a Virtual Display

SMG\$CREATE_VIRTUAL_DISPLAY has an optional argument (**character-set**) for the default character set of the virtual display created. If **character-set** is omitted, Chinese Screen Management Facility uses the SMG\$DEFAULT_CHARACTER_SET logical name to define the default character set. Valid definitions for the SMG\$DEFAULT_CHARACTER_SET logical name are as follows:

Table 2–1 Default Character Set Names

Value	Character Set
ASCII	ASCII
SPEC_GRAPHICS	DEC Special Graphics
HANYU	DEC Hanyu

If an invalid character set name is assigned in SMG\$DEFAULT_CHARACTER_SET logical name, or if the logical name is not defined, the default character set name is unknown.

2.3 Operational Controls

This section describes the Chinese screen management routines that control special modes of operation: minimal update, buffering, and whether or not tabs are used in updating. These modes let you optimize the manner in which information is actually written to the screen. To invoke these modes, you use the SMG\$CONTROL_MODE routine.

Normally, you need not be concerned with these modes; the Chinese Screen Management Facility optimizes output so that characters appear to be displayed on the screen immediately. For some applications, however, you may want to take advantage of these mode settings. The following sections describe these modes of operation.

2.3.1 Minimal Update

By default, the Chinese Screen Management Facility attempts to minimize the number of characters written to the screen by rewriting only the parts of the screen that have changed. However, the Chinese Screen Management Facility also supports nonminimal updating, in which all lines affected by a change are redrawn, beginning at the first changed character and continuing to the end of the line.

2.3.2 Buffering

By default, output operations cause an immediate change on the screen by sending many small, partially filled buffers to the terminal instead of updating the screen when the buffer is full. Minimizing the number of these I/O transactions by enabling buffering mode results in faster program execution.

In buffering mode, the Chinese Screen Management Facility writes the terminal buffer to the screen only when the buffer is full. Thus, several output operations may be performed before the results appear on the screen. Because this delay is not acceptable for many applications, a special routine, SMG\$FLUSH_BUFFER, is provided for use with buffering. SMG\$FLUSH_BUFFER forces the buffer to be written to the terminal whether or not it is full. This routine is useful for an application that can usually accept delayed output but occasionally requires an

Chinese Screen Management Output Operations

2.3 Operational Controls

immediate screen update. Applications that usually need immediate changes on the screen should not enable buffering.

2.3.3 Tabs

Tabs are used for minimal updating. When you are using tabs, you must ensure that the tab stops are set to the Digital default locations. Do not use tabs if you want to be sure that the application will run regardless of the tab settings the user has set on the terminal.

Any tabs that you output to the screen are converted to eight spaces by SMG\$ before being output to the screen. The only exception to this is seen in using SMG\$CREATE_VIRTUAL_DISPLAY with the **display-attributes** argument set to SMG\$M_DISPLAY_CONTROLS. Only in this case is the tab character printed rather than interpreted as eight spaces.

2.4 Batching Output Operations

If you want to construct a complex virtual display that requires several scrolling, cursor positioning, and output operations but do not want the interim steps to be visible, you can batch the output operations. Batching a series of operations to a virtual display lets the application hide the interim steps.

You may also want to construct a complex pasteboard image but have it appear on the screen only after the entire picture is complete. Unpasting and repasting leaves the screen blank during the construction process, so in this case you can batch a series of composition operations and let the screen show only the final effect.

The Chinese Screen Management Facility provides a mechanism for batching a series of operations at both the virtual display level and the pasteboard level. These are described in the following sections.

2.4.1 Display Update Batching

The SMG\$BEGIN_DISPLAY_UPDATE routine causes output operations to a pasted display to be reflected only in the display's buffers. When all operations to the display are finished, the application can call the SMG\$END_DISPLAY_UPDATE routine, which causes the display's buffer to be written to the pasteboard.

The SMG\$BEGIN_DISPLAY_UPDATE and SMG\$END_DISPLAY_UPDATE routines increment and decrement a counter. When this counter's value is zero, output to the virtual display is immediately sent to the pasteboard. When the counter's value is nonzero, output operations are batched; the **display batching level** is equal to the counter's value. Notice that the counter mechanism allows a subroutine to request and turn off batching without disturbing the batching level of the calling program.

2.4.2 Pasteboard Update Batching

You accomplish pasteboard batching by calling the SMG\$BEGIN_PASTEBOARD_UPDATE routine, performing several composition operations, and finally calling the SMG\$END_PASTEBOARD_UPDATE routine. The SMG\$BEGIN_PASTEBOARD_UPDATE routine causes output operations to be reflected only in the pasteboard buffer, not on the physical screen. The SMG\$END_PASTEBOARD_UPDATE routine causes the pasteboard buffer to be written to the physical screen.

Chinese Screen Management Output Operations

2.4 Batching Output Operations

The `SMG$BEGIN_PASTEBOARD_UPDATE` and `SMG$END_PASTEBOARD_UPDATE` routines increment and decrement a counter. When this counter's value is zero, output to the pasteboard is immediately sent to the physical screen. When the counter's value is nonzero, output operations are batched; the **pasteboard batching level** is equal to the value of the counter. Notice that the counter mechanism allows a subroutine to request and turn off batching without disturbing the batching level of the calling program.

Chinese Screen Management Input Operations

This chapter describes the Chinese screen management routines used to perform input from a virtual keyboard. A virtual keyboard is associated with a terminal.

The Chinese Screen Management Facility provides a flexible set of routines for performing input from a terminal. The input routines can be used in conjunction with the output routines, or they can be used by themselves. You establish an input source, called a virtual keyboard, by calling the `SMG$CREATE_VIRTUAL_KEYBOARD` routine. You delete a virtual keyboard by calling the `SMG$DELETE_VIRTUAL_KEYBOARD` routine.

Note

The Chinese Screen Management Facility does not support an RMS file for the input device.

A virtual keyboard is a logical structure for input operations, just as a pasteboard is a logical structure for output operations. The advantage of using virtual keyboards is device independence. When using the screen management input routines, you need not worry about the type of terminal being used. For example, your program need not know which line terminators a particular terminal uses; the Chinese screen management routines map the different terminator character sequences into a uniform set of function codes. (See Section 3.5 for more information about terminator codes.) Virtual keyboards are also important for processing function/keypad keys.

3.1 Obtaining Data from Virtual Keyboards

Data may be obtained from a virtual keyboard in three ways. `SMG$READ_STRING` reads a string composed of characters and a terminator. This flexible routine provides access to many features of the OpenVMS terminal driver. `SMG$READ_COMPOSED_LINE` reads a line composed of ordinary keystrokes and predefined strings associated with keypad and function keys; it provides an easy way to code an interface for command-oriented utilities by providing single-key command capabilities. `SMG$READ_KEYSTROKE` reads one keystroke entered at the keyboard. It reads function/keypad keys as well as alphanumeric keys. All read operations can be aborted by calling the `SMG$CANCEL_INPUT` routine.

The `SMG$READ_COMPOSED_LINE` and `SMG$READ_STRING` routines have an optional argument (**character-set**) for the character set of the text to be input. If **character-set** argument is omitted, Chinese Screen Management Facility uses the `SMG$DEFAULT_CHARACTER_SET` logical name to get the character set of the input text. Refer to Table 2-1 in Section 2.2.14 for the valid definitions of the `SMG$DEFAULT_CHARACTER_SET` logical name. If an invalid character set name is assigned in `SMG$DEFAULT_CHARACTER_SET` logical

Chinese Screen Management Input Operations

3.1 Obtaining Data from Virtual Keyboards

name, or if the logical name is not defined, the character set name of the input text is unknown.

3.2 Setting and Retrieving Virtual Keyboard Characteristics

In the same way that you can retrieve information about and set pasteboard characteristics, the Chinese Screen Management facility also provides routines that let you retrieve and set the characteristics of a virtual keyboard.

The `SMG$GET_KEYBOARD_ATTRIBUTES` routine lets you retrieve requested information about a virtual keyboard. It deposits this information in a user-supplied area called the keyboard information table (KIT). The information returned includes the following:

- The current device characteristics
- The device class
- The size of the recall buffer
- The physical device type
- The first character in the type-ahead buffer
- The terminal width
- The number of characters in the type-ahead buffer

You can use `SMG$SET_KEYPAD_MODE` to set the terminal's numeric keypad to either numeric or applications mode. In applications mode, numeric keypad keys are considered function keys and may be used as terminators. In numeric mode, these keys are equivalent to the corresponding keys on the main keyboard. Note that the terminal must support applications mode, or the call to `SMG$SET_KEYPAD_MODE` will fail.

3.3 Line Composition Using Keypad Keys

In addition to the functions provided by `SMG$READ_STRING`, line composition with keypad keys provides a powerful and flexible tool for applications that have line-oriented commands (for example, utilities that use the Command Definition Utility). (See *OpenVMS Command Definition, Librarian, and Message Utilities Manual* for more information.)

With line composition, you can define certain keys (discussed below) to be equivalent to a string of characters. When you enter a line and press one of these keys, the equivalence string for that key is inserted into the returned command string. For example, if the application defines the key PF2 to have the equivalence string "HELP", then when you press the PF2 key, that command is returned to the application. You can also specify that the equivalence string be echoed; in this case, the string "HELP" is echoed. The recognition of keypad keys and the insertion of the equivalence string are handled automatically by `SMG$READ_COMPOSED_LINE`; the application treats the returned line just as if you had typed the entire line.

Key definitions are placed in a key definition table, which is created by a call to `SMG$CREATE_KEY_TABLE`. Key definitions can be added to and deleted from the table by calls to `SMG$ADD_KEY_DEF` and `SMG$DELETE_KEY_DEF`. Key definitions can also be added by calls to `SMG$DEFINE_KEY` and `SMG$LOAD_KEY_DEFS`; these routines accept a DCL `DEFINE/KEY` command (or a file of these commands). See the description of these routines for more information; see *OpenVMS DCL Dictionary* for an explanation of the `DEFINE/KEY` command.

Chinese Screen Management Input Operations

3.3 Line Composition Using Keypad Keys

The keys that can be defined are function and keypad keys listed in Table 3–1, the control key sequences (Ctrl/A through Ctrl/Z with the exception of Ctrl/M (RETURN)), and line editing keys if line editing is disabled.

A key definition has several attributes. The `TERMINATE` attribute specifies whether the input line is terminated when this key is pressed; the `NOTERMINATE` attribute specifies that more characters and keystrokes may be entered. `TERMINATE` is the default.

The `ECHO` attribute specifies whether the equivalence string is echoed when the key is pressed. `ECHO` is the default.

The `PROTECT` attribute specifies whether this key definition can be changed or deleted once it is defined. `NOPROTECT` is the default.

The remaining attributes are `LOCK_STATE`, `IF_STATE`, and `STATE`. They are described in the following section.

3.4 States

A given key may have many definitions, depending on the value of the current state; the state is used to determine the meaning of the key. For example, if PF1 is defined as setting the state to “GOLD” and if PF2 with `IF_STATE`=“GOLD” is defined as “HELP *”, pressing PF1 and then PF2 would result in “HELP *” being returned as the command line. Note that in this case the PF1 definition would have no equivalence string and would specify the `NOTERMINATE` attribute.

A state name is any string comprising up to 31 alphanumeric characters, and can include the dollar sign (\$) and underscore (_). When a line is being composed from normal keystrokes and equivalence strings, `SMG$READ_COMPOSED_LINE` maintains a string called the *current state name*. Before the first key is pressed, the current state is “DEFAULT”. If you press a key whose definition has specified a value for the `STATE` attribute, the current state is changed to the specified state. Unless you specify the `LOCK_STATE` attribute, the state name reverts to “DEFAULT” after the next defined key is pressed.

3.5 Terminators

A terminator ends a transfer of data from the virtual keyboard. A terminator may be a single character such as a carriage return or Ctrl/Z, a character sequence (escape sequence) generated by pressing a function key on a keyboard, or a condition such as timeout or buffer full.

The terminator is not part of the data read from the virtual keyboard; it is returned to the caller in a separate argument as an integer (unsigned word) value. For single-character terminators, the value is the terminator’s 8-bit character code. Single-character terminator codes are in the range 0 through 255.

Character sequence terminators are returned in a device-independent fashion. The codes are in the form `SMG$K_TRM_keyname` (for example, `SMG$K_TRM_DELETE`). A unique code is assigned to each possible function key on VT220 (and VT200-compatible) terminals. Key codes on other terminals are returned using the code of the equivalent VT220 key. Therefore, the application program need not know which type of terminal is being used; the Chinese screen management routines transparently map the different terminator character sequences into a uniform set of function codes.

Chinese Screen Management Input Operations

3.5 Terminators

Input operations terminated by a condition are indicated by the terminator codes SMG\$K_TRM_CANCELLED, SMG\$K_TRM_TIMEOUT, SMG\$K_TRM_BUFFER_FULL, and SMG\$K_TRM_UNKNOWN.

For calls to SMG\$READ_STRING, the default single terminator characters are all the characters in the range 0 through 31, except backspace (8), horizontal tab (9), line feed (10), vertical tab (11), and form feed (12). Note that these characters make up the default terminator set for the OpenVMS terminal driver. However, any 8-bit character code is potentially a terminator. The set of terminator characters may be changed by calls to SMG\$READ_STRING. For calls to SMG\$READ_COMPOSED_LINE, the only default single terminator characters are the carriage return (13) and Ctrl/Z (26). Changes to the terminator set for SMG\$READ_COMPOSED_LINE are made by key definitions; see the description of line composition in Section 3.3 for more information.

The routine SMG\$NAME_TO_KEYCODE translates the name of a key on the keyboard to its corresponding terminator code, while SMG\$KEYCODE_TO_NAME translates the terminator code to the corresponding name of the key on the keyboard.

Table 3–1 lists the terminator name or condition for each terminator that is not a single character. The table also lists the code and the key legend for each terminator on the different types of terminals supported by the Chinese screen management input routines.

Table 3–1 Terminator Values

Key Name	Value	VT200 and VT300 Series	VT100	VT52
Keypad Keys				
DELETE	SMG\$K_TRM_DELETE	<X>	DELETE	DEL
PF1	SMG\$K_TRM_PF1	PF1	PF1	Blue
PF2	SMG\$K_TRM_PF2	PF2	PF2	Red
PF3	SMG\$K_TRM_PF3	PF3	PF3	Black
PF4	SMG\$K_TRM_PF4	PF4	PF4	
KP0	SMG\$K_TRM_KP0 ¹	0	0	0
KP1	SMG\$K_TRM_KP1 ¹	1	1	1
KP2	SMG\$K_TRM_KP2 ¹	2	2	2
KP3	SMG\$K_TRM_KP3 ¹	3	3	3
KP4	SMG\$K_TRM_KP4 ¹	4	4	4
KP5	SMG\$K_TRM_KP5 ¹	5	5	5
KP6	SMG\$K_TRM_KP6 ¹	6	6	6
KP7	SMG\$K_TRM_KP7 ¹	7	7	7
KP8	SMG\$K_TRM_KP8 ¹	8	8	8
KP9	SMG\$K_TRM_KP9 ¹	9	9	9

¹These are the keys on the numeric keypad, not the main keyboard. These values are used only if the terminal keypad is in applications mode; if the keypad is in numeric mode, the keys are equivalent to the keys with the same legends on the main keyboard. See the description of SMG\$SET_KEYPAD_MODE for more information.

(continued on next page)

Chinese Screen Management Input Operations 3.5 Terminators

Table 3–1 (Cont.) Terminator Values

Key Name	Value	VT200 and VT300 Series	VT100	VT52
ENTER	SMG\$K_TRM_ENTER ²	ENTER	ENTER	ENTER
MINUS	SMG\$K_TRM_MINUS ¹	–	–	
COMMA	SMG\$K_TRM_COMMA ¹	,	,	
PERIOD	SMG\$K_TRM_PERIOD ¹	.	.	
Cursor Positioning Keys				
UP	SMG\$K_TRM_UP	Up arrow	Up arrow	Up arrow
DOWN	SMG\$K_TRM_DOWN	Down arrow	Down arrow	Down arrow
LEFT	SMG\$K_TRM_LEFT	Left arrow	Left arrow	Left arrow
RIGHT	SMG\$K_TRM_RIGHT	Right arrow	Right arrow	Right arrow
Function Keys				
F6	SMG\$K_TRM_F6	F6		
F7	SMG\$K_TRM_F7	F7		
F8	SMG\$K_TRM_F8	F8		
F9	SMG\$K_TRM_F9	F9		
F10	SMG\$K_TRM_F10	F10		
F11	SMG\$K_TRM_F11	F11		
F12	SMG\$K_TRM_F12	F12		
F13	SMG\$K_TRM_F13	F13		
F14	SMG\$K_TRM_F14	F14		
HELP	SMG\$K_TRM_HELP ³	HELP		
DO	SMG\$K_TRM_DO ³	DO		
F17	SMG\$K_TRM_F17	F17		
F18	SMG\$K_TRM_F18	F18		
F19	SMG\$K_TRM_F19	F19		
F20	SMG\$K_TRM_F20	F20		
Editing Keys				
FIND	SMG\$K_TRM_FIND	Find		
INSERT_HERE	SMG\$K_TRM_INSERT_HERE	Insert Here		
REMOVE	SMG\$K_TRM_REMOVE	Remove		
SELECT	SMG\$K_TRM_SELECT	Select		
PREV_SCREEN	SMG\$K_TRM_PREV_SCREEN	Prev Screen		
NEXT_SCREEN	SMG\$K_TRM_NEXT_SCREEN	Next Screen		
Conditions				
CANCELED	SMG\$K_TRM_CANCELLED			

¹These are the keys on the numeric keypad, not the main keyboard. These values are used only if the terminal keypad is in applications mode; if the keypad is in numeric mode, the keys are equivalent to the keys with the same legends on the main keyboard. See the description of SMG\$SET_KEYPAD_MODE for more information.

²If the keypad is in numeric mode, ENTER is equivalent to a carriage return. See the description of SMG\$SET_KEYPAD_MODE for more information.

³HELP and DO are in the F15 and F16 positions on the VT220 keyboard.

(continued on next page)

Chinese Screen Management Input Operations

3.5 Terminators

Table 3–1 (Cont.) Terminator Values

Key Name	Value	VT200 and VT300 Series	VT100	VT52
TIMEOUT	SMG\$K_TRM_TIMEOUT			
BUFFER_FULL	SMG\$K_TRM_BUFFER_FULL			
UNKNOWN	SMG\$K_TRM_UNKNOWN ⁴			

⁴If an unrecognized terminator is received, the value is SMG\$K_TRM_UNKNOWN.

Symbolic definitions of the terminator values are provided in Digital-supplied symbol libraries named \$SMGDEF (for example, in a MACRO program you would issue a call to \$SMGDEF to extract these definitions). The symbol names are of the form SMG\$K_TRM_keyname, where **keyname** is the key name given in Table 3–1. For terminator codes 1 through 26 which correspond to the control sequences Ctrl/A through Ctrl/Z, the key names are CTRLA for Ctrl/A, CTRLB for Ctrl/B, and so on. The following synonyms are also defined:

Symbol	Synonym
SMG\$K_TRM_BS	SMG\$K_TRM_CTRLH
SMG\$K_TRM_HT	SMG\$K_TRM_CTRLI
SMG\$K_TRM_LF	SMG\$K_TRM_CTRLJ
SMG\$K_TRM_CR	SMG\$K_TRM_CTRLM
SMG\$K_TRM_E1	SMG\$K_TRM_FIND
SMG\$K_TRM_E2	SMG\$K_TRM_INSERT_HERE
SMG\$K_TRM_E3	SMG\$K_TRM_REMOVE
SMG\$K_TRM_E4	SMG\$K_TRM_SELECT
SMG\$K_TRM_E5	SMG\$K_TRM_PREV_SCREEN
SMG\$K_TRM_E6	SMG\$K_TRM_NEXTSCREEN
SMG\$K_TRM_F15	HELP
SMG\$K_TRM_F16	DO

3.6 Line Recall and the Recall Buffer

The Chinese Screen Management Facility allows you to access and change the contents of the application recall buffer. By default, the recall buffer stores the previous 20 commands or data lines entered by the user to the application.

The SMG\$RETURN_INPUT_LINE routine lets you request a particular line from the recall buffer. You can either specify the appropriate line number for the line to be recalled, or you can specify a match string. If you use a match string, SMG\$RETURN_INPUT_LINE searches for and returns the line that matches the specified string. This routine is intended to aid in the implementation of a DCL-style RECALL command.

The SMG\$REPLACE_INPUT_LINE routine lets you replace the specified line or lines in the recall buffer with the specified string. The remaining lines of the recall buffer are deleted. This routine is intended to aid in processing line continuations.

3.7 Interaction of Input and Output

SMG\$READ_COMPOSED_LINE, SMG\$READ_KEYSTROKE, and SMG\$READ_STRING accept an optional **display-id** argument. If a **display-id** is supplied, it designates the virtual display in which the input operation should occur. By specifying **display-id**, you enable the Chinese Screen Management Facility to remain aware of the changes caused by character echoing. If you omit **display-id**, the Chinese Screen Management Facility assumes that Chinese screen management output is not being used.

Note that if the **display-id** argument is specified for any one of the above-mentioned input routines, then the length of the prompt string plus the input is limited to the number of columns in the display or, where specified, to the maximum number of characters to be read. (In the case of SMG\$READ_KEYSTROKE, this restriction applies only to the length of the prompt string.)

Advanced Screen Management Features

The Chinese Screen Management Facility provides several advanced features to:

- Trap asynchronous events
- Create and execute commands in a subprocess
- Move the physical cursor
- Clean up at exit

The following sections describe these features.

Note

The Chinese Screen Management Facility assumes that it has complete control of the terminal. Applications should not mix calls to SMG\$ with calls to other screen products such as DEC GKS or VAX FMS.

SMG\$ is not reentrant. Therefore, unpredictable results can occur when multiple processes call SMG\$ routines to perform operations on the same terminal. If your application contains multiple processes, one process should call the SMG\$ routines. Other processes should send messages to the calling process when screen modifications are necessary. The calling process can then call the appropriate SMG\$ routines.

4.1 Asynchronous Events

There are three types of asynchronous events that can disrupt the screen image:

- Broadcast messages
- Unsolicited input
- Out-of-band asynchronous system traps

The following sections explain how to control these actions.

Note

The Chinese Screen Management Facility is not AST reentrant. Therefore, the caller of the SMG\$ routines described in this chapter is responsible for any synchronization needed.

Advanced Screen Management Features

4.1 Asynchronous Events

4.1.1 Broadcast Messages

Normally, broadcast messages (for example, MAIL notifications or operator messages) can appear on the terminal screen at any time, destroying or distorting the screen image. The `SMG$SET_BROADCAST_TRAPPING` routine lets you trap messages broadcast to the specified terminal (pasteboard) and in addition, lets you specify an AST routine to be called whenever a broadcast message is trapped. The AST routine you supply can access the broadcast message by calling the `SMG$GET_BROADCAST_MESSAGE` routine.

Whether or not you specify an AST routine in the call to `SMG$SET_BROADCAST_TRAPPING`, you can check for the receipt of a broadcast message at any time by calling `SMG$GET_BROADCAST_MESSAGE`.

4.1.2 Unsolicited Input

The `SMG$ENABLE_UNSOLICITED_INPUT` routine detects the presence of unsolicited input. Note that this routine does not read any input characters; it merely calls an AST routine to notify the application that it should issue a read operation with `SMG$READ_COMPOSED_LINE`, `SMG$READ_KEYSTROKE`, or `SMG$READ_STRING`. It is up to you to read the unsolicited input.

4.1.3 Out-of-Band ASTs

The `SMG$SET_OUT_OF_BAND_ASTS` routine provides a way to trap out-of-band characters such as Ctrl/Y, Ctrl/C, and Ctrl/O. This routine lets you specify which characters are to be treated as out-of-band characters and also lets you specify an AST routine to be called when one of these characters is typed.

4.2 Subprocesses

The Chinese Screen Management Facility lets you use a subprocess to execute DCL commands from an application. Only one subprocess is allowed per virtual display.

4.2.1 Creating a Subprocess

You create a subprocess with the `SMG$CREATE_SUBPROCESS` routine. This routine creates a DCL subprocess and associates it with a virtual display you specify. The subprocess is initialized with the `SET NOVERIFY` and `SET NOON` DCL commands. The creating process requires an available `BYTLM` value of at least 5000 and an available `PRCLM` value of at least 1. The Screen Management Facility checks to make sure that you have sufficient resources before creating the subprocess.

4.2.2 Deleting a Subprocess

When you are done executing subprocess commands, you can delete the subprocess with the `SMG$DELETE_SUBPROCESS` routine. If you exit without first calling this routine, the Chinese Screen Management Facility includes an exit handler that deletes the subprocess for you. It is important to note, however, that under some circumstances these facility-supplied exit handlers are not executed. In that case, you must delete the subprocess with the following commands:

```
$ SHOW PROCESS/SUB
$ STOP/IDENT=xxxx
```

4.2.3 Executing Commands in a Subprocess

The `SMG$EXECUTE_COMMAND` routine lets you execute a specified command in the created subprocess. If commands are being buffered, `SMG$EXECUTE_COMMAND` returns control after the specified command is buffered. The `AST` routine that you specify is invoked with the command status when the command completes execution. If commands are not being buffered, `SMG$EXECUTE_COMMAND` waits until the command has completed execution before returning the status of the command.

When you specify the command string to be executed, you must specify a “\$” as the first character of any DCL command. The Screen Management Facility assumes that any command string that does not begin with a “\$” is input data for the previous command. The commands and their outputs are displayed on the specified virtual display as they are executed. Note that the commands `SPAWN`, `GOTO`, and `LOGOUT` are illegal to use as command strings and generate unpredictable results.

It is also important to note that since I/O is performed by means of mailboxes and not through the terminal driver, single-character commands such as `Ctrl/C`, `Ctrl/Y`, `Ctrl/Z`, and so on have no effect. Use `SMG$M_SEND_EOF` as the **flags** parameter in order to pass a `Ctrl/Z` to the subprocess.

4.3 Moving the Cursor

The Chinese Screen Management Facility lets you move the cursor to a specified location on the physical screen. It does so through the `SMG$SET_PHYSICAL_CURSOR` routine. However, if you attempt to move the cursor to a pasteboard position outside the screen boundaries, an error is returned.

4.4 Exit Handler

The Chinese Screen Management Facility supplies an exit handler, which is invoked before image termination. This handler deletes all pasteboards and virtual keyboards associated with the current image and resets the terminal characteristics. The Chinese Screen Management Facility’s exit handler may or may not be invoked before any user-supplied exit handlers. Therefore, you should not delete pasteboards or virtual displays from inside a user-supplied exit handler because they may already have been deleted by the Chinese Screen Management Facility exit handler and their identifiers deassigned.

Support for Third Party Terminals

This chapter describes SMG\$ support for *foreign* terminals. A foreign terminal is any terminal for which the device type is not one of the Digital terminals recognized by OpenVMS, or any terminal on which the ANSI_CRT characteristic is not set. This support is used by the Chinese Screen Management Facility but it can also be used by application programs that need to do their own I/O to foreign terminals. Thus, if you use the Chinese Screen Management Facility, you need to concern yourself only with the definition of foreign terminal capabilities, and not with the details of calling the foreign terminal routines directly. Further, you need to define only a few terminal capabilities (“set absolute cursor position,” “erase to end of display,” and “erase to end of line”) in order for the Screen Management Facility to effectively control the terminal screen. However, the routines used by the Chinese Screen Management Facility are presented here to allow you to do your own I/O to foreign terminals.

The support begins with a source file named TERMTABLE.TXT, which contains a list of terminal names and their associated capabilities. This file is processed by the SYS\$SYSTEM:SMGBLDTRM.EXE program to create an image file called TERMTABLE.EXE. The following sections describe the creation and processing of the TERMTABLE database.

Note that the TERMTABLE support is used by the Chinese Screen Management Facility for all terminals, including Digital terminals. The definitions for Digital terminals are included in a file named SYS\$SYSTEM:SMGTERMS.TXT, which is provided as part of the Chinese Screen Management Facility. The examples in the following section show you how to use the foreign terminal package to define Digital terminals, because most users are familiar with them. Note that you should not create your own definitions for Digital terminals, nor should you modify the definitions in SYS\$SYSTEM:SMGTERMS.TXT.

5.1 TERMTABLE Interface Routines

TERMTABLE.EXE is a database containing information about any number of different types of terminals. You extract information from this database in three steps:

1. Provide the terminal name to the database.
2. Retrieve the information about that terminal type (this step might be repeated any number of times).
3. End access to the database.

Note that when using Chinese screen management routines to perform I/O to foreign terminals, you need only create the proper TERMTABLE entries for the foreign terminals you use. The steps listed above are necessary only when your program is doing I/O directly to foreign terminals.

Support for Third Party Terminals

5.1 TERMTABLE Interface Routines

The first step can be performed in either of two ways. You can either pass a string that contains a terminal name (for example, “VT100”) to the `SMG$INIT_TERM_TABLE` routine, or you can pass a value returned by the OpenVMS system service `$GETDVI` to the `SMG$INIT_TERM_TABLE_BY_TYPE` routine. The returned value may be a symbolic terminal type (for example, `TT$_VT100` or `TT$_VT52`) or a value assigned by the `SMG$` foreign terminal routines to designate a particular foreign terminal.

The second step requires that you call the `SMG$GET_TERM_DATA` routine. This routine extracts a command string (for example, an escape sequence) from `TERMTABLE` and stores it in a buffer you provide. It is then your responsibility to write the command string to the terminal. Note that it may be necessary to call `SMG$GET_TERM_DATA` many times; each time you receive the command sequence you can perform a different operation. Note that you should also call `SMG$GET_TERM_DATA` each time you want to use a capability string that requires a substitution or arithmetic operation for an argument. However, you may want to save the static capability strings in your program’s local storage. These static capability strings can be retrieved once and used any number of times.

The third step is optional; it merely frees the virtual memory used to access the information in the database.

Note that the `DCL SET` and `SHOW TERMINAL` commands recognize any name defined in `TERMTABLE`, as well as the current set of valid OpenVMS terminal names. If you use the `SET TERMINAL/DEVICE=name` command to specify a terminal that is unknown to the OpenVMS operating system, the `TERMTABLE` database is searched for the named terminal.

Two routines are provided to obtain the address of a specific terminal definition. `SMG$INIT_TERM_TABLE` accepts a terminal name as input; `SMG$INIT_TERM_TABLE_BY_TYPE` accepts a device type as input. Each maps to a specific terminal entry in the `TERMTABLE.EXE` section. These routines return this identifier to the caller for use in future calls.

`SMG$GET_TERM_DATA` accepts the identifier of the compiled `TERMTABLE` database and a request code. The request code is used as an index into the data to retrieve the appropriate escape sequence. Some sequences are static; they do not contain any variable information and are simply copied into the caller’s buffer. Variable sequences, which include a `!` or `%` directive, cause additional processing to take place. An example of a variable sequence is the VT300-series set cursor command: the required binary row and column numbers must be converted to ASCII for the set cursor sequence. `SMG$GET_TERM_DATA` uses the optional input arguments to do the conversion before copying the sequence to the caller’s buffer.

Note that if you do not provide any optional input arguments to `SMG$GET_TERM_DATA`, it uses a default of 1 for each argument that the capability requires. Nevertheless, you cannot supply some of the optional arguments and accept the default for others—you must supply all or none of them. `SMG$GET_NUMERIC_DATA` provides a simplified interface for users who wish to obtain numeric or Boolean data only.

When all terminal I/O has been completed, `SMG$DEL_TERM_TABLE` can be called to release the virtual memory used. This routine is useful only if you do not need `TERMTABLE` for the duration of your program; releasing virtual memory may make it available for reuse by your program.

Support for Third Party Terminals

5.1 TERMTABLE Interface Routines

A skeleton TERMTABLE.TXT is supplied in SYS\$SYSTEM:. SMGTERMS.TXT, which defines Digital terminals, is also provided. The skeleton TERMTABLE.TXT uses the REQUIRE directive to include the separate source file SMGTERMS.TXT. Thus, only third party terminals are actually defined in the TERMTABLE.TXT source file.

Note that SMGTERMS.TXT should not be modified by users.

5.2 Capability Fields

If you have a foreign (third party) terminal, the Chinese Screen Management Facility does not know what your terminal can and cannot do—in other words, what the terminal's capabilities are. Capability fields let you tell the Chinese Screen Management Facility what capabilities are supported for your foreign terminal. These fields let SMG\$ use your terminal capabilities rather than emulate common terminal functions, which in turn improves SMG\$'s performance.

Three types of capability fields are allowed in a TERMTABLE entry:

- Boolean
- Numeric
- String

The following sections describe these capability fields in detail.

Functions that are common to most terminals have been chosen as possible fields; not all functions of all terminal types are represented. (Specifically, there is no support for block mode, graphics, or typesetting composition functions.) Screen-oriented applications should be planned around typical terminal functions, and should not depend on features that exist on only one or a few models.

For applications that must support an unusual terminal, some generic capability names are reserved for user definition. Names of the form PRIVATE_BOO_n, PRIVATE_NUM_n, and PRIVATE_STR_n, where *n* is a number from 1 to 10, may be included as user-defined terminal definitions and returned by the TERMTABLE interface routines. Since meanings are assigned by the user, private capabilities will vary between applications. Sites running several applications must guard against multiple definitions for a single private capability. (For example, you should include separate terminal entries for a terminal that requires PRIVATE_STR_1 to mean two different things, depending on the application program being run.) In general, you should not have to use private capabilities.

The following characters are used as delimiters in capability fields.

Delimiter	Meaning
!	Begins a comment
=	Separates a capability field name from its value
,	Separates capability fields
"	Delimits strings

Support for Third Party Terminals

5.2 Capability Fields

5.2.1 Boolean Capability Fields

Boolean capabilities are either present or not present for a given terminal.

The format for a Boolean capability field is as follows:

BOOLEAN {boolean-capability = binary-digit} [...]

Following are the meanings of the elements:

boolean-capability One of the capability fields listed in Table 5–1

binary-digit Either 1 or 0

Table 5–1 lists these Boolean capability fields.

Table 5–1 Boolean Capabilities

OpenVMS Name	Used by SMG	Description
ADVANCED_VIDEO	N	If set, the terminal has advanced video attributes and is capable of 132-column mode operation
ANSI_CRT	N	If set, terminal conforms to ANSI CRT programming standards
AUTO_MARGIN	N	If set, terminal has automatic margins
BACKSPACE	Y	If set, terminal can backspace with Ctrl/H
BLOCK_MODE	N	If set, terminal can perform block mode transmission, local editing, and field protection
CURSOR_REPORT_ANSI	N	If set, terminal uses the ANSI sequence to report the current cursor location
DEC_CRT	N	If set, terminal conforms to Digital VT100 family standards
DEC_CRT_2	N	If set, terminal conforms to Digital VT200 family standards
DEC_CRT_3	N	If set, terminal conforms to Digital VT300 family standards
DEC_CRT_4	N	If set, terminal conforms to Digital VT400 family standards
DEC_CRT_5	N	If set, terminal conforms to Digital VT500 family standards
EDIT	N	If set, terminal can perform ANSI-defined advanced editing functions
EIGHT_BIT	N	If set, terminal uses 8-bit ASCII character code
FULLDUP	N	If set, terminal operation mode is full-duplex (half-duplex if not set)
IGNORE_NEWLINE	N	If set, terminal ignores a newline after a wrap
INSERT_MODE_NULLS	N	If set, insert mode distinguishes nulls on display
LOWERCASE	N	If set, terminal has both uppercase and lowercase letters
NO_ERASE	N	If set, standout (bolded) characters are not erased by writing over them

(continued on next page)

Table 5–1 (Cont.) Boolean Capabilities

OpenVMS Name	Used by SMG	Description
NO_SCROLL	N	If set, terminal is not capable of scrolling
OVERSTRIKE	N	If set, terminal is capable of overstriking
PHYSICAL_FF	N	If set, terminal can accept form feeds (If not set, terminal driver must translate form feeds to multiple line feeds)
PHYSICAL_TABS	N	If set, terminal has hardware tabs (Note that these tabs may need to be set with an initialization string)
PRINTER_PORT	N	If set, terminal has a printer port available
PRIVATE_BOO_1 to 10	N	If set, these fields denote user-defined capabilities 1 through 10
REGIS	N	If set, terminal understands ReGIS graphics commands
SCOPE	N	If set, terminal is a video terminal
SET_CURSOR_COL_ROW	Y	If set, terminal uses column/row addressing
SIXEL_GRAPHICS	N	If set, terminal can display graphics using the ReGIS-defined SIXEL graphics protocol
SOFT_CHARACTERS	N	If set, terminal can load a user-defined character set
UNDERLINE	N	If set, terminal has underlining capability (but not overstrike)

For example, the following TERMTABLE entry describes two characteristics of a VT300 series terminal:

```
NAME = "VT300_series"
    BOOLEAN
        ansi_crt = 1,          dec_crt = 1
```

This entry specifies that the terminal conforms to ANSI CRT programming standards and to Digital VT300-series standards.

5.2.2 Numeric Capability Fields

Numeric capabilities are those that take a numeric argument, for example, the number of columns on the terminal screen.

The format for a numeric capability field is as follows:

```
NUMERIC {numeric-capability = value} [...]
```

Following are the meanings of the elements:

numeric-capability	One of the capability fields listed in Table 5–2
value	The value for the specified numeric capability

Support for Third Party Terminals

5.2 Capability Fields

Table 5–2 lists numeric capabilities.

Table 5–2 Numeric Capabilities

OpenVMS Name	Used by SMG	Description
COLUMNS	Y	Specifies the number of columns in a line
CR_FILL	N	Specifies the number of fill characters needed after a carriage return
LF_FILL	N	Specifies the number of fill characters needed after a line feed
FRAME	N	Controls the number of data bits expected by the terminal driver for every character that is input or output (value must be between 5 and 8, inclusive)
NUMBER_FN_KEYS	N	Specifies the number of function keys available
PRIVATE_NUM_1 to 10	N	If set, these fields denote user-defined capabilities 1 through 10
ROWS	N	Specifies the number of rows on the screen
WIDE_SCREEN_COLUMNS	Y	Specifies the number of columns available in wide mode

For example, the following TERMTABLE entry describes two characteristics of a VT300-series terminal:

```
NAME = "VT300_series"  
    NUMERIC  
        rows = 24,          columns = 80
```

5.2.3 String Capability Fields

String capability fields provide several features. They let you do the following:

- Supply alternate characters for line drawing
- Provide icons so that your program can display carriage control characters (for example, form feeds) instead of executing them
- Supply the character sequences that cause a given operation (for example, ERASE_TO_END_OF_LINE) to be performed on any type of terminal
- Specify the character strings returned by special keys (for example, function keys) on a given terminal
- Specify strings having a maximum length of 255 characters (the maximum length of a string depends on its complexity, but in no case can it be longer than 255)

Table 5–3 lists string capabilities.

Support for Third Party Terminals 5.2 Capability Fields

Table 5–3 String Capabilities

Name	Used by SMG	Description
BEGIN_ALTERNATE_CHAR	N	Begins alternate character set
BEGIN_AUTOPRINT_MODE	N	Begins autoprint mode
BEGIN_AUTOREPEAT_MODE	N	Begins autorepeat mode
BEGIN_AUTOWRAP_MODE	N	Begins autowrap mode
BEGIN_BLINK	Y	Begins blinking characters
BEGIN_BOLD	Y	Begins bolded characters
BEGIN_DELETE_MODE	N	Begins delete mode
BEGIN_INSERT_MODE	N	Begins insert mode
BEGIN_LINE_DRAWING_CHAR	Y	Begins using line-drawing character set
BEGIN_NORMAL_RENDITION	Y	Begins using normal video attributes
BEGIN_REVERSE	Y	Begins reverse video characters
BEGIN_UNDERSCORE	Y	Begins underscored characters
BOTTOM_T_CHAR	Y	Displays line-drawing character <i>bottom t</i>
CLEAR_TAB	N	Clears tab at current column
CR_GRAPHIC	Y	Defines character to indicate a carriage return when control characters are being represented rather than executed
CROSS_CHAR	Y	Defines character to represent the intersection of perpendicular lines
CURSOR_DOWN	N	Moves cursor <i>n</i> lines down (does not cause scrolling)
CURSOR_LEFT	N	Moves cursor <i>n</i> positions to the left
CURSOR_NEXT_LINE	N	Accepts an argument <i>n</i> and moves the cursor to the first position in the <i>n</i> th following line
CURSOR_POSITION_REPORT	N	Reports the active position using two arguments
CURSOR_PRECEDING_LINE	N	Accepts an argument <i>n</i> and moves the cursor to the first position in the <i>n</i> th preceding line
CURSOR_RIGHT	N	Accepts an argument <i>n</i> and moves the cursor <i>n</i> positions to the right
CURSOR_UP	N	Accepts an argument <i>n</i> and moves cursor up <i>n</i> lines (does not cause scrolling)
DARK_SCREEN	Y	Makes screen background color dark (normal video)
DELETE_CHAR	N	Accepts an argument <i>n</i> and deletes <i>n</i> characters
DELETE_LINE	N	Accepts an argument <i>n</i> and deletes <i>n</i> lines
DEVICE_ATTRIBUTES	N	Terminal's response to a "What are you?" sequence
DOUBLE_HIGH_BOTTOM	Y	Changes line to double height bottom half
DOUBLE_HIGH_TOP	Y	Changes line to double height top half

(continued on next page)

Support for Third Party Terminals

5.2 Capability Fields

Table 5–3 (Cont.) String Capabilities

Name	Used by SMG	Description
DOUBLE_WIDE	Y	Changes line to double width
END_ALTERNATE_CHAR	N	Ends alternate character set
END_AUTOPRINT_MODE	N	Ends autoprint mode
END_AUTOREPEAT_MODE	N	Ends autorepeat mode
END_AUTOWRAP_MODE	N	Ends autowrap mode
END_BLINK	N	Ends blinking characters
END_BOLD	N	Ends bolding mode
END_DELETE_MODE	N	Ends delete mode
END_INSERT_MODE	N	Ends insert mode
END_LINE_DRAWING_CHAR	Y	Ends line-drawing characters
END_REVERSE	N	Ends reverse video characters
END_UNDERSCORE	N	Ends underscore
ERASE_DISPLAY_TO_CURSOR	N	Erases display to virtual cursor position
ERASE_LINE_TO_CURSOR	N	Erases line to virtual cursor position
ERASE_TO_END_DISPLAY	N	Erases to end of display
ERASE_TO_END_LINE	Y	Erases to end of line
ERASE_WHOLE_DISPLAY	Y	Erases whole display
ERASE_WHOLE_LINE	N	Erases whole line
ERROR_ICON	Y	Defines character that indicates an error
FF_GRAPHIC	Y	Uses this character to indicate a form feed when control characters are displayed rather than executed
HOME	Y	Defines home cursor
HORIZONTAL_BAR	Y	Displays line-drawing character <i>horizontal bar</i>
HT_GRAPHIC	Y	Uses this character to indicate a horizontal tab when control characters are displayed rather than executed
INDEX	N	Moves the cursor down one line without changing the column position (contents of the screen scroll up if necessary)
INIT_STRING	Y	Defines terminal initialization string
INSERT_CHAR	N	Accepts an argument <i>n</i> and inserts <i>n</i> characters
INSERT_LINE	N	Accepts an argument <i>n</i> and inserts <i>n</i> lines
INSERT_PAD	N	Accepts an argument <i>n</i> and inserts <i>n</i> pad characters after character inserted
KEY_0	Y	Returned by keypad 0 in applications mode
KEY_1	Y	Returned by keypad 1 in applications mode

(continued on next page)

Support for Third Party Terminals 5.2 Capability Fields

Table 5–3 (Cont.) String Capabilities

Name	Used by SMG	Description
KEY_2	Y	Returned by keypad 2 in applications mode
KEY_3	Y	Returned by keypad 3 in applications mode
KEY_4	Y	Returned by keypad 4 in applications mode
KEY_5	Y	Returned by keypad 5 in applications mode
KEY_6	Y	Returned by keypad 6 in applications mode
KEY_7	Y	Returned by keypad 7 in applications mode
KEY_8	Y	Returned by keypad 8 in applications mode
KEY_9	Y	Returned by keypad 9 in applications mode
KEY_BACKSPACE	N	Returned by backspace key
KEY_COMMA	Y	Returned by keypad comma key
KEY_DOWN_ARROW	Y	Returned by down arrow key
KEY_E1	Y	Returned by E1 (editing key 1)
KEY_E2	Y	Returned by E2 (editing key 2)
KEY_E3	Y	Returned by E3 (editing key 3)
KEY_E4	Y	Returned by E4 (editing key 4)
KEY_E5	Y	Returned by E5 (editing key 5)
KEY_E6	Y	Returned by E6 (editing key 6)
KEY_ENTER (k)	Y	Returned by keypad enter key
KEY_F1	Y	Returned by F1 (function key 1)
.		.
.		.
.		.
KEY_F20	Y	Returned by F20 (function key 20)
KEY_LABEL_F1	N	Legend on F1 (function key 1)
.		.
.		.
.		.
KEY_LABEL_F20	N	Legend on F20 (function key 20)
KEY_LEFT_ARROW	Y	Returned by left arrow key
KEY_MINUS	Y	Returned by keypad minus key
KEY_PERIOD	Y	Returned by keypad period key
KEY_PF1	Y	Returned by PF1 key
KEY_PF2	Y	Returned by PF2 key
KEY_PF3	Y	Returned by PF3 key
KEY_PF4	Y	Returned by PF4 key
KEY_RIGHT_ARROW	Y	Returned by right arrow key
KEY_UP_ARROW	Y	Returned by up arrow key
LEFT_T_CHAR	Y	Displays line-drawing character <i>left t</i>

(continued on next page)

Support for Third Party Terminals

5.2 Capability Fields

Table 5–3 (Cont.) String Capabilities

Name	Used by SMG	Description
LF_GRAPHIC	Y	Uses this character to indicate a line feed when control characters are displayed rather than executed
LIGHT_SCREEN	Y	Makes screen background color light (reverse video)
LOWER_LEFT_CORNER	Y	Displays line-drawing character <i>lower left corner</i>
LOWER_RIGHT_CORNER	Y	Displays line-drawing character <i>lower right corner</i>
NAME	Y	Defines terminal name; must be the first field in the entry
NEWLINE_CHAR	N	Defines new-line character
NEXT_LINE	N	Displays next line
NO_PRINTER	N	Defines no attached printer status
PAD_CHAR	N	Defines pad character (if other than null)
PRINT_SCREEN	N	Prints contents of screen
PRINTER_READY	N	Defines printer ready status
PRINTER_NOT_READY	N	Defines printer not ready status
PRIVATE_STR_1	N	User-defined capability 1
.	.	.
.	.	.
PRIVATE_STR_10	N	User-defined capability 10
REQUEST_CURSOR_POSITION	N	Requests the active cursor position
REQUEST_PRINTER_STATUS	N	Requests status of attached printer
RESTORE_CURSOR	N	Restores cursor to previously saved position
REVERSE_INDEX	N	Moves the cursor to the same column on the preceding line (contents of the screen scroll down if necessary)
RIGHT_T_CHAR	Y	Displays line-drawing character <i>right t</i>
SAVE_CURSOR	N	Saves cursor position
SCROLL_FORWARD	N	Accepts an argument <i>n</i> and scrolls forward <i>n</i> lines
SCROLL_REVERSE	Y	Accepts an argument <i>n</i> and scrolls backward <i>n</i> lines
SEL_ERASE_TO_END_DISPLAY	N	Selectively erases from cursor to end of display (does not change attributes)
SEL_ERASE_TO_END_LINE	N	Selectively erases from cursor to end of line (does not change attributes)
SEL_ERASE_WHOLE_DISPLAY	N	Selectively erases entire display (does not change attributes)

(continued on next page)

Support for Third Party Terminals 5.2 Capability Fields

Table 5–3 (Cont.) String Capabilities

Name	Used by SMG	Description
SEL_ERASE_WHOLE_LINE	N	Selectively erases entire line (does not change attributes)
SET_APPLICATION_KEYPAD	Y	Begins applications keypad mode
SET_CHAR_NOT_SEL_ERASE	N	Designates all subsequent characters as not selectively erasable
SET_CHAR_SEL_ERASE	N	Designates all subsequent characters as selectively erasable
SET_CURSOR_ABS	Y	Directs cursor addressing (accepts row and column arguments)
SET_CURSOR_OFF	Y	Sets cursor to invisible
SET_CURSOR_ON	Y	Sets cursor to visible
SET_JUMP_SCROLL	Y	Sets scrolling to jump scroll
SET_NUMERIC_KEYPAD	Y	Ends keypad applications mode (resumes numeric mode)
SET_ORIGIN_ABSOLUTE	N	Allows cursor positioning outside current scrolling region
SET_ORIGIN_RELATIVE	N	Prohibits cursor positioning outside current scrolling region
SET_PRINTER_OUTPUT	N	Sends output to printer port rather than screen
SET_SCREEN_OUTPUT	N	Sends output to terminal screen
SET_SCROLL_REGION	Y	Sets scrolling region (accepts as arguments top margin and bottom margin)
SET_SMOOTH_SCROLL	Y	Sets scrolling to smooth scroll
SET_TAB	N	Sets tab at current column
SINGLE_HIGH	Y	Changes this line to single height, single width
TAB_CHAR	N	Defines tab character (other than Ctrl/I or tab with padding); note that this field should be used only for non-ASCII terminals
TOP_T_CHAR	Y	Displays line-drawing character <i>top t</i>
TRUNCATION_ICON	Y	Defines the character that indicates overflow characters were truncated

(continued on next page)

Support for Third Party Terminals

5.2 Capability Fields

Table 5–3 (Cont.) String Capabilities

Name	Used by SMG	Description
UNDERLINE_CHAR	N	Underlines a character
UPPER_LEFT_CORNER	Y	Displays line-drawing character <i>upper left corner</i>
UPPER_RIGHT_CORNER	Y	Displays line-drawing character <i>upper right corner</i>
VERTICAL_BAR	Y	Displays line-drawing character <i>vertical bar</i>
VT_GRAPHIC	Y	Defines the character that indicates a vertical tab when control characters are displayed rather than executed
WIDTH_NARROW	Y	Sets terminal width to narrow (usually 80 columns)
WIDTH_WIDE	Y	Sets terminal width to wide (usually 132 columns)

Table 5–4 lists the `STRING_2` capabilities. In order to construct one of the user-defined renditions `SMG$M_USER1` through `SMG$M_USER8`, you must provide an appropriate definition in the file `TERMTABLE.TXT` using `STRING_2` capabilities.

Table 5–4 String_2 Capabilities

Name	Used by SMG	Description
BEGIN_STATUS_LINE	Y	Begins output to hardware status line
BEGIN_USER1	Y	Begins first user-defined attribute
BEGIN_USER2	Y	Begins second user-defined attribute
BEGIN_USER3	Y	Begins third user-defined attribute
BEGIN_USER4	Y	Begins fourth user-defined attribute
BEGIN_USER5	Y	Begins fifth user-defined attribute
BEGIN_USER6	Y	Begins sixth user-defined attribute
BEGIN_USER7	Y	Begins seventh user-defined attribute
BEGIN_USER8	Y	Begins eighth user-defined attribute
END_STATUS_LINE	Y	Ends output to hardware status line
END_USER1	N	Ends first user-defined attribute
END_USER2	N	Ends second user-defined attribute
END_USER3	N	Ends third user-defined attribute
END_USER4	N	Ends fourth user-defined attribute
END_USER5	N	Ends fifth user-defined attribute
END_USER6	N	Ends sixth user-defined attribute

(continued on next page)

Table 5–4 (Cont.) String_2 Capabilities

Name	Used by SMG	Description
END_USER7	N	Ends seventh user-defined attribute
END_USERS8	N	Ends eighth user-defined attribute
BLACK_SCREEN	Y	Makes screen background color black
BLUE_SCREEN	Y	Makes screen background color blue
CYAN_SCREEN	Y	Makes screen background color cyan (green-blue)
GREEN_SCREEN	Y	Makes screen background color green
MAGENTA_SCREEN	Y	Makes screen background color magenta
RED_SCREEN	Y	Makes screen background color red
WHITE_SCREEN	Y	Makes screen background color white
YELLOW_SCREEN	Y	Makes screen background color yellow
USER1_SCREEN	Y	User-defined background color
USER2_SCREEN	Y	User-defined background color

Because string capability fields often include nonprinting characters, the following substitutions are used to make it easy to insert these characters in a capability string. Use the special character to represent the nonprinting character.

Special Character	Nonprinting Character	Meaning
\$	ESCAPE	ASCII decimal value 27
^	CONTROL	Control
&	CSI	ASCII decimal value 155
@	SS3	ASCII decimal value 143

Thus to create a capability string that contains an escape character, you simply insert a dollar sign at that position. To create a capability string that contains a control character, prefix the character with a circumflex (^). For example:

```
NAME = "VT300_series"
.
.
.
STRING
    begin_alternate_char = "^N",
    end_alternate_char = "^O",
    erase_whole_display = "${2J}"
.
.
.
END
```

If you want to use a character in a capability string and have its normal ASCII value, place an underscore in front of it. (For example, “_” results in a single dollar sign rather than an underscore followed by an escape character). The following characters must be preceded by an underscore in order to be treated as normal ASCII text:

Support for Third Party Terminals

5.2 Capability Fields

Ampersand	&
Apostrophe	'
At sign	@
Quotation marks	"
Circumflex	^
Dollar sign	\$
Exclamation point	!
Left parenthesis	(
Underscore	_

Note that the Chinese Screen Management Facility automatically invokes the graphics mode needed to display the line-drawing character set (for example, the *bottom_t_char*, *top_t_char*, and so on). However, if you call the foreign terminal routines directly, you are responsible for invoking the required graphics mode.

Padding (for example, with null characters) must sometimes be added to a terminal command to allow the terminal sufficient time to execute the command. The amount of padding needed depends on the terminal and the baud rate. The pad character capability field is included only for future expansion and has no function in this release; padding is the responsibility of the user.

When the foreign terminal support routines are called directly, many of the string capability fields use arguments whose values must be specified at run time. Further, some arguments also require that arithmetic operations be performed when values are substituted for arguments. The following sections describe argument substitution and arithmetic operations.

5.2.4 Argument Substitution

It is frequently necessary to substitute values in a terminal command string. For example, setting a scrolling region or moving the cursor ten columns to the right requires the run-time substitution of a value; these values cannot be stored in the TERMTABLE terminal definition. TERMTABLE provides for string substitution by accepting !UL, an \$FAO style directive. The !UL directive signifies that a value is to be inserted at that point: the TERMTABLE interface routine is to accept an unsigned longword and convert it to ASCII digits before substituting it in the capability field string (and thus in the returned command string). For example:

```
NAME = "VT300_series"
.
.
.
STRING
set_cursor_abs = "$[!UL;!ULH"
.
.
.
END
```

The string defined for the SET_CURSOR_ABS function must have values substituted for the two !UL directives; these values specify the row and column number at which to set the cursor. You specify these run-time arguments as an optional longword vector argument to the SMG\$GET_TERM_DATA routine. The first entry in the vector contains the number of arguments that follow. Thus, the

first entry is 2, the second entry is the desired row number, and the third entry is the desired column number. The SMG\$GET_TERM_DATA routine converts the first optional data item (the second item in the vector) to ASCII digits and substitutes an ASCII value for the first !UL directive; it converts the second optional data item and substitutes it for the second !UL directive, and so on.

5.2.5 Arithmetic Operations

In addition to argument substitution, terminal command sequences may also require arithmetic operations. To perform an argument substitution and arithmetic operation, the TERMTABLE entry requires a different scheme than for argument conversion and substitution.

To perform both argument substitution and arithmetic operations, you use an opening parenthesis, a percent sign (to indicate the point of substitution), an arithmetic operator, an operand, and a closing parenthesis. For example:

```
NAME = "VT52"
.
.
.
STRING
set_cursor_abs = "$Y(%1+31)(%2+31)"
.
.
.
END
```

This example shows the string that directly positions the cursor on a VT52, where a bias must be added to the row and column numbers. Values to be substituted in the expression are passed with the SMG\$GET_TERM_DATA routine, in the same way as for argument substitution alone. The percent sign is always followed by an integer that indicates the order in which arguments should be substituted.

The following table summarizes the characters used in arithmetic operations:

Character	Meaning
(Beginning of arithmetic expression
%n	Substitute <i>n</i> th user argument
+	Arithmetic addition operator
-	Arithmetic subtraction operator
*	Arithmetic multiplication operator
/	Arithmetic division operator
)	End of arithmetic expression

Note that longword integers should be sufficient to express screen coordinates. Expressions are evaluated from left to right; there is no operator precedence.

Spaces between items are not significant; they may be used wherever desired to improve readability. Capability strings are limited to 128 bytes in length.

Support for Third Party Terminals

5.3 Creating an OpenVMS Terminal Capabilities File

5.3 Creating an OpenVMS Terminal Capabilities File

The source code for the database is an ASCII file named TERMTABLE.TXT. This file contains an entry for each type of terminal. Each entry lists a terminal's capabilities and other device-specific information, such as initialization sequences and screen size; a TERMTABLE entry can span more than one record in the file. A terminal definition can be added by editing the TERMTABLE.TXT file; TERMTABLE.TXT must then be reprocessed by running SYS\$SYSTEM:SMGBLDTRM.EXE.

TERMTABLE.TXT can be created with any text editor. A TERMTABLE entry consists of a terminal name, followed by any number of capability fields and their values (see Section 5.2 for more information about capability fields). Although TERMTABLE.TXT must be formatted for compilation, capability names are descriptive and can be easily understood. Terminal names must be unique; for example, if more than one definition is needed for a foreign terminal, then a second name must be used.

When a TERMTABLE routine first searches for a terminal entry, it tries to find TERMTABLE.EXE in the area logically named TERM\$TABLOC. If the specified terminal entry is not found there, the routine then searches for TERMTABLE.EXE in SYS\$SYSTEM. If you want to use a terminal definition that differs from the system definition for that terminal, you can create a private copy of TERMTABLE.TXT and TERMTABLE.EXE. You can then define a single terminal with a definition that is different from the one in SYS\$SYSTEM:TERMTABLE.EXE and still use the rest of the standard system definitions.

The format of a TERMTABLE entry is as follows:

```
NAME = "terminal-name" capability-field [...] END
```

Note that the TERMTABLE.TXT file allows you to include REQUIRE directives. The REQUIRE directive lets you include separate source files in the TERMTABLE.TXT file. Its format is as follows:

```
REQUIRE "filespec"
```

In the above format, "filespec" is a valid OpenVMS file specification.

5.4 Examples

```
!  
!      Private versions of Digital terminal definitions  
!  
NAME = 'myvt300'  
  
  BOOLEAN  
  ansi_crt = 1,          dec_crt = 1  
  
  NUMERIC  
  rows = 24,            columns = 80,  
  wide_screen_columns = 132  
  
  STRING  
  begin_alternate_char = "^N",  
  end_alternate_char = "^O",  
  erase_whole_display = "$[2J",  
  init_string = "$_(B",  
  set_cursor_abs = "$[!UL;!ULH"  
  
END  
  
NAME = "MYVT52"
```

Support for Third Party Terminals

5.4 Examples

```
BOOLEAN
ansi_crt = 0,          dec_crt = 1

NUMERIC
rows = 24,            columns = 80,
wide_screen_columns = 80

STRING
begin_alternate_char = "$F"
end_alternate_char = "$G",
erase_whole_display = "$Y(32)(32)$J", !position to 1,1; then erase
set_cursor_abs = "$Y(%1+31)(%2+31)"

END
```

For the set cursor sequence listed for a VT300-series (MYVT300), the string returned depends on the values provided in the argument vector supplied with the call to the SMG\$GET_TERM_DATA routine. For example, to position the cursor to row 3 and column 12, you supply these longword values as the second and third entries in the vector (the first entry is the number of values that follow). The SMG\$GET_TERM_DATA routine converts these longword values into their ASCII values and inserts the converted values into the string returned at the point of the respective !UL directives.

For the set cursor sequence listed for a VT52 (MYVT52), the string returned depends not on argument substitution, but on an arithmetic operation (because the VT52 requires biasing). The arithmetic operator is used to add 31(decimal) to the row and column numbers supplied in entries 2 and 3 of the argument vector for the SMG\$GET_TERM_DATA routine.

The INIT_STRING field in MYVT300 is included to point out that the parenthesis is normally treated as a special character indicating an arithmetic expression. A parenthesis must be preceded by an underscore in order to be interpreted as a normal text character. Thus the string "\$_(B)" yields ESC(B, a command that designates the ASCII character set into G0.

The ERASE_WHOLE_DISPLAY sequence for MYVT52 shows that it may be necessary to combine sequences in order to provide a certain function. The VT52 does not have a command that erases the entire screen. However, you can erase the entire screen by homing the cursor and then using the command that erases from the current position to the end of the screen.

The following BASIC example program uses the LIB\$GETDVI routine to ascertain the type of terminal associated with SYS\$OUTPUT. The program then uses the foreign terminal routines to place the cursor at the twelfth screen line and to erase to the end of the screen. Note that the program detects whether these capabilities are available for the terminal and displays an error message if they are not.

```
10 ! Program to call the Termtable interface routines
!
! This program will set the cursor to row 12 column 1,
! and erase to the bottom of the screen. If the cursor
! positioning or erasing to the end of the screen
! capabilities are not defined, a message will be output.

%INCLUDE "$SMGTRMPTR" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"
OPTION TYPE = EXPLICIT, SIZE = INTEGER LONG

EXTERNAL INTEGER FUNCTION SYS$ASSIGN, SYS$DASSGN, SYS$QIOW
EXTERNAL INTEGER FUNCTION LIB$GETDVI, LIB$GET_EF, LIB$FREE_EF
EXTERNAL INTEGER FUNCTION SMG$INIT_TERM_TABLE_BY_TYPE, SMG$GET_TERM_DATA

EXTERNAL INTEGER CONSTANT IO$_WRITEVBLK, DVI$_DEVTYPE
```

Support for Third Party Terminals

5.4 Examples

```

COMMON (buf) STRING Data_buffer = 20          ! buffer to hold terminal data
DECLARE INTEGER Sys_status,                   &
              Chan,                           &
              Term_type,                      &
              Term_table_addr,                &
              Arg_vector (2),                 &
              Ret_len,                        &
              Event_flag

! Assign a channel for LIB$GETDVI and SYS$QIOW.
Sys_status = SYS$ASSIGN ('SYS$OUTPUT', Chan, , , )
IF (Sys_status AND 1%) = 0%
THEN
  PRINT "Error from SYS$ASSIGN : ";Sys_status
  GOTO Done
END IF

! Get the terminal type.
Sys_status = LIB$GETDVI (DVI$_DEVTYPE ! request item code           &
                        ,Chan        ! channel assigned to SYS$OUTPUT &
                        ,             ! omit device name             &
                        ,Term_type)  ! place to return type

IF (Sys_status AND 1%) = 0%
THEN
  PRINT "Error from LIB$GETDVI : ";Sys_status
  GOTO Done
END IF

! Get the definition for the type of terminal we are running on.
Sys_status = SMG$INIT_TERM_TABLE_BY_TYPE (Term_type, Term_table_addr)
IF (Sys_status AND 1%) = 0%
THEN
  PRINT "Error getting terminal definition : ";Sys_status
  GOTO Done
END IF

! Get the sequence to position the cursor to 12,1
Arg_vector (0) = 2%          ! number of args to follow
Arg_vector (1) = 12%         ! row number
Arg_vector (2) = 1%          ! column number

Sys_status = SMG$GET_TERM_DATA                                     &
            ( Term_table_addr ! addr of terminal definition &
              ,SMG$K_SET_CURSOR_ABS ! request code           &
              ,20%             ! max buffer length          &
              ,Ret_len         ! length of sequence returned &
              ,Data_buffer BY REF ! buffer to hold sequence &
              ,Arg_vector (0) ) ! optional vector with
                                ! row and column numbers

IF (Sys_status AND 1%) = 0%
THEN
  PRINT "Error getting cursor sequence : ";Sys_status
  GOTO Done
END IF

IF Ret_len = 0%
THEN
  PRINT "Cursor sequence not available"
  GOTO Done
END IF

! Get a unique event flag number

```

Support for Third Party Terminals 5.4 Examples

```

Sys_status = LIB$GET_EF (Event_flag)
IF (Sys_status AND 1%) = 0%
THEN
    PRINT "Unable to allocate an event flag"
    GOTO Done
END IF

! Output the cursor sequence to the terminal.
Sys_status = SYS$QIOW ( Event_flag BY VALUE      ! event flag number &
                    ,Chan BY VALUE              ! channel number     &
                    ,IO$_WRITEVBLK BY VALUE     ! function code     &
                    , , ,                       ! no iosb,          &
                    , , ,                       ! ast routine,      &
                    , , ,                       ! or argument       &
                    ,Data_buffer BY REF         ! buffer to output  &
                    ,Ret_len BY VALUE           ! bytes returned    &
                    , , , , )                   ! null arguments

IF (Sys_status AND 1%) = 0%
THEN
    PRINT "Error from SYS$QIOW : ";Sys_status
    GOTO Done
END IF

! Get the sequence to erase from current cursor to end of screen.
Sys_status = SMG$GET_TERM_DATA                                     &
            ( Term_table_addr ! addr of terminal definition      &
            ,SMG$_ERASE_TO_END_DISPLAY ! request code         &
            ,20%                  ! max buffer length    &
            ,Ret_len                ! bytes returned       &
            ,Data_buffer BY REF)    ! buffer for sequence

IF (Sys_status AND 1%) = 0%
THEN
    PRINT "Error getting erase sequence : ";Sys_status
    GOTO Done
END IF

IF Ret_len = 0%
THEN
    PRINT "Erase sequence not available"
    GOTO Done
END IF

! Output the erase sequence to the terminal.
Sys_status = SYS$QIOW (Event_flag BY VALUE      ! event flag number &
                    ,Chan BY VALUE              ! channel number     &
                    ,IO$_WRITEVBLK BY VALUE     ! function code value &
                    , , ,                       ! no iosb,          &
                    , , ,                       ! ast routine,      &
                    , , ,                       ! or argument       &
                    ,Data_buffer BY REF         ! buffer to output  &
                    ,Ret_len BY VALUE           ! bytes in buffer    &
                    , , , , )                   ! null arguments

IF (Sys_status AND 1%) = 0%
THEN
    PRINT "Error from SYS$QIOW : ";Sys_status
    GOTO Done
END IF

! Deassign the channel.
Sys_status = SYS$DASSGN (Chan BY VALUE)

```

Support for Third Party Terminals

5.4 Examples

```
IF (Sys_status AND 1%) = 0%
THEN
  PRINT "Error from SYS$DASSGN : ";Sys_status
  GOTO Done
END IF

! Deallocate event flag so other programs can use it.

Sys_status = LIB$FREE_EF (Event_flag)

IF (Sys_status AND 1%) = 0%
THEN
  PRINT "Unable to deallocate event flag"
  GOTO Done
END IF

32767 Done: END
```

5.5 Creating TERMTABLE.EXE

Accessing an ASCII file for each screen program is inefficient because the ASCII text must be processed as binary information before it can be returned as a string ready for the terminal. To avoid paying the price of this processing at the start of every image, TERMTABLE.TXT is “precompiled” into the required binary format. A screen application then gets its terminal sequences from the precompiled binary capabilities file.

You compile TERMTABLE.TXT by running the SYS\$SYSTEM:SMGBLDTRM.EXE program. This utility accepts TERMTABLE.TXT as an input file and creates TERMTABLE.EXE as an output file on the device and directory pointed to by the logical TERM\$TABLOC.

The compiled terminal capabilities are stored as a table in a file which is mapped as a permanent global section. Thus, user programs map to the global section rather than having their own copies of the capabilities data.

If a user compiles a private TERMTABLE.TXT from his or her own directory, the interface routines access it by mapping it as a temporary section. TERMTABLE interface routines look for a definition in the temporary section before looking in the system’s permanent global section.

Note that system managers may want to coordinate terminal definitions so that nonstandard definitions are confined to a user’s private area.

Most users do not have the privilege to create a permanent global section. A short program, SYS\$SYSTEM:SMGMAPTRM.EXE, that maps the compiled TERMTABLE as a global section, is part of the standard OpenVMS startup procedure. In order to map an updated TERMTABLE.EXE as the global section, the existing global section must first be deleted. Deleting the global section while the system is active may cause a user’s program to fail; therefore the system must be rebooted in order to make an updated TERMTABLE.EXE the default.

To reduce compiling time and the size of the resulting global section, the terminal definitions in SYS\$SYSTEM:TERMTABLE.TXT should be kept to a minimum. Only the types of terminals that are actually attached to the computer system should be defined.

5.6 Capability Fields Used by Chinese Screen Management

The tables in Section 5.2 show whether or not the Chinese Screen Management Facility can request a particular capability string. Some functions, such as wide characters or line drawing, will be requested only if the user calls the Chinese screen management routines which output wide text or draw lines. If all you want to do is write normal text to the screen, only the following set of fields needs to be defined:

Essential Capabilities

- NAME
- SET_CURSOR_ABS

SMG operation is more efficient if the following optional capabilities are also provided:

- ERASE_TO_END_DISPLAY
- ERASE_TO_END_LINE
- SET_SCROLLING_REGION

If you do not include ERASE_TO_END_DISPLAY, ERASE_TO_END_LINE, or SET_SCROLLING_REGION, the Chinese Screen Management Facility will insert blanks to perform these functions. However, inserting blanks is a slower operation. Similarly, hardware scrolling also improves output speed; if scrolling is not available, the Chinese Screen Management Facility must rewrite the entire screen.

The Chinese Screen Management Facility uses the ASCII character set. If your terminal has a line-drawing character set, you should define the line drawing characters (*bottom_t_char*, *horizontal_bar*, and so forth). If line-drawing characters are not defined, SMG uses normal ASCII characters to draw borders.

The Chinese Screen Management Facility also relies on the terminal characteristics maintained by the terminal driver. You can change these characteristics with the DCL SET TERMINAL command. For example, if you type SET TERMINAL/NOTAB, then the Chinese Screen Management Facility does not send tabs to the terminal.

5.7 Input Support for Foreign Terminals

A **foreign terminal** is any terminal for which the device type is not one of the standard Digital terminals recognized by the OpenVMS operating system, or any terminal on which the ANSI_CRT characteristic is not set.

When you use a Digital (or ANSI) terminal, typing a special key such as a function key or a keypad key sends an escape sequence (as defined by the ANSI standard) to the OpenVMS terminal driver. The OpenVMS terminal driver understands this ANSI standard and interprets the escape sequence according to this standard. Thus, the OpenVMS terminal driver knows how long the escape sequence is and what characters are allowed in which positions in that sequence.

The OpenVMS terminal driver does not echo any of the printing characters from the sequence because those characters are interpreted with special meaning as part of the escape sequence. Normal keys would be echoed unless the TRM\$M_TM_NOECHO modifier was specified.

Support for Third Party Terminals

5.7 Input Support for Foreign Terminals

The OpenVMS terminal driver returns to SMG\$ the sequence, the length of the sequence, and the number of characters entered before the function key was pressed. SMG\$ determines which key was pressed by comparing the sequence and its length against the list of key definitions for that particular terminal in `TERMTABLE.EXE`. This code is returned to the user in the format `SMG$K_TRM_xxx`, where `xxx` is used to specify the particular key.

When you press a special key such as a function key or a keypad key on a foreign terminal, a non-ANSI sequence is sent to the OpenVMS terminal driver. If this sequence starts with a control character, the OpenVMS terminal driver interprets this character as a terminator. (By default all control characters are terminators unless you use a terminator mask to specify otherwise.) The terminal driver then stops reading characters and returns to SMG\$ the character, a length of 1, and the number of characters entered before the function key was pressed.

SMG\$ looks at the returned character. If it is a control character, SMG\$ looks in the type-ahead buffer for the next characters of the sequence. If there are characters in the type-ahead buffer, SMG\$ will read one character from the type-ahead buffer, append it to the control sequence it has already, and check this new sequence against the list of key definitions for this terminal in `TERMTABLE.EXE` to determine which key was pressed. If the sequence is not matched, the next character is read from the type-ahead buffer. This continues until a match is found or the type-ahead buffer is empty. Since the terminal driver does not know about this sequence, any printable characters in the sequence are echoed by the terminal driver unless the `noecho` modifier was specified by the user. Because SMG\$ does not know what characters will make up this sequence, it disables line editing in order to allow the actual characters that make up the sequence to be returned to SMG\$.

Terminals whose special keys send a sequence that does not start with a control character are not supported by SMG\$ for input. Changing the terminator mask to exclude the control character that starts the function key sequence is not supported. In addition, the performance of a foreign terminal doing input will not match that of a Digital terminal doing the same input since SMG\$ must parse the sequence in place of the OpenVMS terminal driver.

5.8 Support for SET and SHOW TERMINAL Commands

The DCL `SET TERMINAL` command is the mechanism for setting your terminal to conform to a `TERMTABLE` definition. `SET TERMINAL` causes the following three fields to be retrieved from the `TERMTABLE` database and set for your terminal.

1. Terminal type—A signed integer assigned by the system and associated with a particular device type
2. Width—The number of columns on the physical screen
3. Page size—The number of rows on the screen

In addition, if the corresponding Boolean capability is set in the terminal definition, the following flags are set:

- `ADVANCED_VIDEO`
- `ANSI_CRT`
- `BLOCK_MODE`
- `DEC_CRT`

Support for Third Party Terminals

5.8 Support for SET and SHOW TERMINAL Commands

- EDIT
- EIGHT_BIT
- FORM
- FULLDUP
- LOWERCASE
- REGIS
- SCOPE
- SIXEL_GRAPHICS
- SOFT_CHARACTERS
- TAB

If any of these fields is missing from your definition, the previous setting for that characteristic is retained; SET TERMINAL does not try to set that characteristic for your terminal. You should include all of the above capabilities in your definitions to avoid unpredictable settings.

SET TERMINAL will operate as it always has for known terminals such as the VT300 series, VT200 series, VT100, and VT52. When SET TERMINAL encounters an unknown device name, it will search TERMTABLE for a definition with that name. Notice that your definitions must have names other than the names that SET TERMINAL currently recognizes. The terminals currently recognized are listed as follows:

LA12	LQP02	VT125
LA34	VT05	VT131
LA36	VT52	VT132
LA38	VT55	VT200-SERIES
LA100	VT100	VT300-SERIES
LA120	VT101	FT1 through FT8
Unknown	VT102	

If SET TERMINAL finds the device name in its own internal tables, it does not search the TERMTABLE database.

Since the SET TERMINAL command recognizes only the first 15 characters of a device name, you may want to limit your terminal names to 15 characters.

The SET TERMINAL/DEVICE=name command causes the TERMTABLE database to be searched for the named terminal, if that terminal is unknown to the OpenVMS operating system. SET TERMINAL/DEVICE=name then sets various terminal characteristics, as shown in the following table, based on the presence of these capabilities in the TERMTABLE database.

Capability Field	Terminal Characteristic
LOWERCASE	LOWERCASE
PHYSICAL_TABS	TABS
SCOPE	SCOPE
EIGHT_BIT	EIGHTBIT

Support for Third Party Terminals

5.8 Support for SET and SHOW TERMINAL Commands

Capability Field	Terminal Characteristic
PHYSICAL_FF	FORM
FULLDUP	FULLDUP
SIXEL_GRAPHICS	SIXEL
SOFT_CHARACTERS	SOFT
ANSI_CRT	ANSI_CRT
REGIS	REGIS
BLOCK_MODE	BLOCK
ADVANCED_VIDEO	AVO
EDIT_MODE	EDIT
DEC_CRT	DEC_CRT

The SET TERMINAL/DEVICE_TYPE= format must be used with TERMTABLE terminals. SET TERMINAL/name is an old format that works for a small set of device names and is maintained only for compatibility with previous versions of the OpenVMS operating system.

Using Chinese Screen Management Routines to Develop New Programs

This chapter discusses some recommended methods for using the Chinese Screen Management Facility for developing new programs. It is important to note that Chinese screen management routines are not AST reentrant.

Note

The image of Chinese Screen Management Facility is HSMGSHR.EXE at SYS\$SHARE. Asian character set literals (SMG\$C_HANYU and SMG\$C_UNKNOWN) have been defined in STARLET since OpenVMS V5.4. Users can also extract those values from STARLET.MLB.

There are two ways in which an application can call Chinese screen management routines.

- Directly

Applications that call the Chinese Screen Management Facility directly already using pasteboards and virtual displays.

- Indirectly

This kind of application does not use the Chinese Screen Management Facility directly, but may use it in the course of invoking other routines.

As time goes on, and more and more callable routines may use the Screen Management Facility to produce their output, it becomes more difficult to determine whether your application is in this category.

In either case, the calling routine is likely at some point to call a subsystem so that the subsystem can write data to the screen.

At some later point, the terminal user will want to remove the subsystem-specific display. However, if the subsystem created and used a virtual display to display the data, the display identifier is not available to the calling program and therefore the calling program cannot remove it. Furthermore, unless the calling program is a direct user of the Screen Management Facility, the screen's pasteboard identifier is also not available to it.

The solution is to require that all callable routines that use the Chinese Screen Management Facility, directly or indirectly, have an (optional) input argument for the **pasteboard-id** and an (optional) output argument for the virtual **display-id**. Passing the pasteboard and display identifiers lets you avoid accumulating subsystem-specific data on the screen that cannot be removed by the calling program. These guidelines are developed as follows:

Using Chinese Screen Management Routines to Develop New Programs

- If the **pasteboard-id** argument is provided by the calling program, then
 - The called program should not create a pasteboard of its own.
 - The called program must deliver all of its output to the pasteboard supplied by the calling program; that is, the called program may paste its displays only to the pasteboard specified by **pasteboard-id**.
 - The called program can delete any virtual displays it created by calling `SMG$DELETE_VIRTUAL_DISPLAY`, but it must not delete the pasteboard.

Note that the called program should not simply call the `SMG$UNPASTE_VIRTUAL_DISPLAY` routine with the expectation that this virtual display can be reused in a later invocation. Since the called program and the calling program are sharing a pasteboard, the calling program may use the `SMG$POP_VIRTUAL_DISPLAY` routine to delete all displays created by the called program.
 - The called program must pass the **pasteboard-id** on to any routines it in turn calls. Thus all output is directed to the specified pasteboard.
- If the **pasteboard-id** argument is not provided by the calling program, then
 - The called program must create a pasteboard on its own. The called program may allocate any physical device for the pasteboard, unless specifically directed to a particular device by some other mechanism.

The called program must check the status of the `SMG$CREATE_PASTEBOARD` call to see whether it created a unique pasteboard identifier or whether it received the pasteboard identifier of an already existing pasteboard. If the pasteboard already exists, the called program must not delete the pasteboard.
 - If the called routine creates a pasteboard and in turn calls subroutines that may use pasteboards, it should pass the **pasteboard-id** to the subroutines.
 - The called program may clean up by using the `SMG$UNPASTE_VIRTUAL_DISPLAY` routine, and the displays can be saved for reuse on a subsequent invocation if such a call seems likely. Note, however, that the `SMG$UNPASTE_VIRTUAL_DISPLAY` routine should be used only if the called program creates its own pasteboard, because in this case the calling program cannot delete the virtual displays created by the called program.
- If the virtual **display-id** argument is provided by the calling program, then the calling program must clean up any virtual displays created by the called program. The called program must return to the calling program the identifier of the first virtual display pasted. The calling program can then remove this and all later-pasted virtual displays by calling the `SMG$POP_VIRTUAL_DISPLAY` routine.
- If the virtual **display-id** argument is not provided by the calling program, the called program must remove all the virtual displays it pastes to the pasteboard.

Using Chinese Screen Management Routines to Develop New Programs

By adhering to the following guidelines, you can develop your application in a modular fashion:

- Calling programs control the pasteboard on which information is pasted. Pasteboard identifiers flow downward in a hierarchy, with each routine using the **pasteboard-id** provided by the caller and passing it along to subroutines.
- If a calling program supplies a virtual **display-id** argument to be filled in by the called program, then the calling program assumes responsibility for cleaning up any displays created by the called program. The called program passes back the **display-id** of the first virtual display pasted so that the calling program can remove this and all later-pasted displays by calling the `SMG$POP_VIRTUAL_DISPLAY` routine.
- Virtual displays are created (and pasted) in the routine where they are needed. If the calling program does not supply a **display-id** argument, then displays are unpasted and/or deleted in the routine that created them.

6.1 Calling Routines That Do Not Use the Chinese Screen Management Facility

A different situation exists if you call a subroutine (or subsystem) that writes to the screen without using the Chinese Screen Management Facility. When the Chinese Screen Management Facility is bypassed (that is, when text is placed on the screen outside Chinese screen management's control), problems result when an attempt is made to perform a screen update.

For this reason, the Chinese Screen Management Facility provides two routines for turning over the screen (or a part of it) temporarily to a program that does not use Chinese screen management, and for restoring the screen to its previous state after control is returned from the non-SMG\$ routine. These routines are `SMG$SAVE_PHYSICAL_SCREEN` and `SMG$RESTORE_PHYSICAL_SCREEN`.

Before you call a routine that performs non-SMG\$ I/O to the screen, you should call the `SMG$SAVE_PHYSICAL_SCREEN` routine, specifying what part of the screen is to be turned over to the non-SMG\$ routine. `SMG$SAVE_PHYSICAL_SCREEN` erases the specified area, sets the terminal's physical scrolling region to this area, and sets the physical cursor to row 1, column 1 of the area. If the non-SMG\$ code does only sequential input and output (that is, if it does no direct cursor addressing) its output will be confined to the specified area of the screen.

When control is returned from the non-SMG\$ routine, you simply call `SMG$RESTORE_PHYSICAL_SCREEN`, which restores the screen image as it was before the call to `SMG$SAVE_PHYSICAL_SCREEN`.

Examples of Calling SMG\$ Routines

This chapter contains examples demonstrating how to call the routine `SMG$READ_KEYSTROKE` from all major programming languages. (Note that not all of the languages represented in these examples are available on OpenVMS AXP systems.) Other SMG\$ routines such as `SMG$CREATE_VIRTUAL_DISPLAY`, `SMG$CREATE_PASTEBOARD`, `SMG$CREATE_VIRTUAL_KEYBOARD`, `SMG$PASTE_VIRTUAL_DISPLAY`, and `SMG$PUT_LINE` are also used throughout these examples.

Example 7–1 demonstrates the use of `SMG$READ_KEYSTROKE` from a VAX Ada program. This program also uses `SMG$CREATE_VIRTUAL_DISPLAY`, `SMG$CREATE_PASTEBOARD`, `SMG$CREATE_VIRTUAL_KEYBOARD`, `SMG$PASTE_VIRTUAL_DISPLAY`, and `SMG$PUT_LINE`.

Example 7–1 Using SMG\$ Routines in VAX Ada

```
with SYSTEM, CONDITION_HANDLING; use SYSTEM;
package SMG is -- declarations of SMG$ routines used

  procedure CREATE_VIRTUAL_DISPLAY (
    STATUS: out CONDITION_HANDLING.COND_VALUE_TYPE;
    ROWS, COLUMNS: INTEGER;
    DISPLAY_ID: out INTEGER;
    DISPLAY_ATTRIBUTES, VIDEO_ATTRIBUTES, CHAR_SET: UNSIGNED_LONGWORD
      := UNSIGNED_LONGWORD'NULL_PARAMETER);
  pragma INTERFACE (SMG, CREATE_VIRTUAL_DISPLAY);
  pragma IMPORT_VALUED_PROCEDURE
    (CREATE_VIRTUAL_DISPLAY, "SMG$CREATE_VIRTUAL_DISPLAY");

  procedure CREATE_PASTEBOARD (
    STATUS: out CONDITION_HANDLING.COND_VALUE_TYPE;
    PASTEBOARD_ID: out INTEGER;
    OUTPUT_DEVICE: STRING := STRING'NULL_PARAMETER;
    ROWS, COLUMNS: INTEGER := INTEGER'NULL_PARAMETER;
    PRESERVE_SCREEN_FLAG: BOOLEAN := BOOLEAN'NULL_PARAMETER);
  pragma INTERFACE (SMG, CREATE_PASTEBOARD);
  pragma IMPORT_VALUED_PROCEDURE
    (CREATE_PASTEBOARD, "SMG$CREATE_PASTEBOARD");

  procedure CREATE_VIRTUAL_KEYBOARD (
    STATUS: out CONDITION_HANDLING.COND_VALUE_TYPE;
    KEYBOARD_ID: out INTEGER;
    FILESPEC, DEFAULT_FILESPEC, RESULTANT_FILESPEC: STRING
      := STRING'NULL_PARAMETER);
  pragma INTERFACE (SMG, CREATE_VIRTUAL_KEYBOARD);
  pragma IMPORT_VALUED_PROCEDURE
    (CREATE_VIRTUAL_KEYBOARD, "SMG$CREATE_VIRTUAL_KEYBOARD");
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-1 (Cont.) Using SMG\$ Routines in VAX Ada

```
procedure PASTE_VIRTUAL_DISPLAY (
    STATUS: out CONDITION_HANDLING.COND_VALUE_TYPE;
    DISPLAY_ID, PASTEBOARD_ID: INTEGER;
    ROW, COLUMN: INTEGER);
pragma INTERFACE (SMG, PASTE_VIRTUAL_DISPLAY);
pragma IMPORT_VALUED_PROCEDURE
    (PASTE_VIRTUAL_DISPLAY, "SMG$PASTE_VIRTUAL_DISPLAY");

procedure READ_KEYSTROKE (
    STATUS: out CONDITION_HANDLING.COND_VALUE_TYPE;
    KEYBOARD_ID: INTEGER;
    TERMINATOR_CODE: out UNSIGNED_WORD;
    PROMPT: STRING := STRING'NULL_PARAMETER;
    TIMEOUT, DISPLAY_ID: INTEGER := INTEGER'NULL_PARAMETER);
pragma INTERFACE (SMG, READ_KEYSTROKE);
pragma IMPORT_VALUED_PROCEDURE
    (READ_KEYSTROKE, "SMG$READ_KEYSTROKE");

procedure PUT_LINE (
    STATUS: out CONDITION_HANDLING.COND_VALUE_TYPE;
    DISPLAY_ID: INTEGER;
    TEXT: STRING;
    LINE_ADVANCE: INTEGER := INTEGER'NULL_PARAMETER;
    RENDITION_SET, RENDITION_COMPLEMENT: UNSIGNED_LONGWORD
        := UNSIGNED_LONGWORD'NULL_PARAMETER;
    WRAP_FLAG: BOOLEAN := BOOLEAN'NULL_PARAMETER;
    CHAR_SET: UNSIGNED_LONGWORD := UNSIGNED_LONGWORD'NULL_PARAMETER);
pragma INTERFACE (SMG, PUT_LINE);
pragma IMPORT_VALUED_PROCEDURE
    (PUT_LINE, "SMG$PUT_LINE");
end SMG;

-- This routine demonstrates the use of the SMG$ routines, in particular
-- SMG$READ_KEYSTROKE.

with SMG, STARLET, CONDITION_HANDLING, SYSTEM;
procedure SMG_DEMO is
    STATUS: CONDITION_HANDLING.COND_VALUE_TYPE;
    PASTEBOARD_1, DISPLAY_1, KEYBOARD_1: INTEGER;
    TERMINATOR: SYSTEM.UNSIGNED_WORD;
begin
    -- Create virtual display, pasteboard and virtual keyboard.
    SMG.CREATE_VIRTUAL_DISPLAY (STATUS, ROWS => 7, COLUMNS => 60,
        DISPLAY_ID => DISPLAY_1,
        DISPLAY_ATTRIBUTES => STARLET.SMG_M_BORDER);
    SMG.CREATE_PASTEBOARD (STATUS, PASTEBOARD_ID => PASTEBOARD_1);
    SMG.CREATE_VIRTUAL_KEYBOARD (STATUS, KEYBOARD_ID => KEYBOARD_1);

    -- Paste the virtual display at row 3, column 9.
    SMG.PASTE_VIRTUAL_DISPLAY (STATUS, DISPLAY_ID => DISPLAY_1,
        PASTEBOARD_ID => PASTEBOARD_1, ROW => 3, COLUMN => 9);

    -- Write the instructions to the virtual display.
    SMG.PUT_LINE (STATUS, DISPLAY_ID => DISPLAY_1,
        TEXT => "Enter the character K after the >> prompt.");
    SMG.PUT_LINE (STATUS, DISPLAY_ID => DISPLAY_1,
        TEXT => "This character will not be echoed as you type it.");
    SMG.PUT_LINE (STATUS, DISPLAY_ID => DISPLAY_1,
        TEXT => "The terminal character equivalent of K is displayed.");
    SMG.PUT_LINE (STATUS, DISPLAY_ID => DISPLAY_1,
        TEXT => " ");
end;
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-1 (Cont.) Using SMG\$ Routines in VAX Ada

```
-- Read the keystroke from the virtual keyboard.
SMG.READ_KEYSTROKE (STATUS, KEYBOARD_ID => KEYBOARD_1,
  DISPLAY_ID => DISPLAY_1,
  TERMINATOR_CODE => TERMINATOR, PROMPT => ">>");
-- Display the decimal value of the terminator code.
SMG.PUT_LINE (STATUS, DISPLAY_ID => DISPLAY_1,
  TEXT => " ");
SMG.PUT_LINE (STATUS, DISPLAY_ID => DISPLAY_1,
  TEXT => "TERMINAL CHARACTER IS " &
  SYSTEM.UNSIGNED_WORD' IMAGE(TERMINATOR));
end SMG_DEMO;
```

Examples of Calling SMG\$ Routines

Example 7-2 uses SMG\$READ_KEYSTROKE to read a keystroke from the terminal. This BASIC program also uses SMG\$CREATE_VIRTUAL_KEYBOARD and SMG\$DELETE_VIRTUAL_KEYBOARD.

Example 7-2 Using SMG\$ Routines in VAX BASIC

```
1      OPTION TYPE=EXPLICIT

      !+
      ! This routine demonstrates the use of SMG$READ_KEYSTROKE to read
      ! a keystroke from the terminal.
      !
      ! Build this program using the following commands.
      !
      !$ BASIC READ_KEY
      !$ CREATE SMGDEF.MAR
      !      .TITLE  SMGDEF - Define SMG$ constants
      !      .Ident  /1-000/
      !
      !      $SMGDEF GLOBAL
      !
      !      .END
      !$ MACRO SMGDEF
      !$ LINK READ_KEY,SMGDEF
      !
      !-

      DECLARE LONG KB_ID, RET_STATUS, TERM_CODE, I, TIMER
      EXTERNAL SUB LIB$SIGNAL( LONG BY VALUE )
      EXTERNAL SUB LIB$STOP( LONG BY VALUE )
      EXTERNAL LONG CONSTANT SS$_TIMEOUT
      EXTERNAL LONG CONSTANT SMG$K_TRM_PF1
      EXTERNAL LONG CONSTANT SMG$K_TRM_PERIOD
      EXTERNAL LONG CONSTANT SMG$K_TRM_UP
      EXTERNAL LONG CONSTANT SMG$K_TRM_RIGHT
      EXTERNAL LONG CONSTANT SMG$K_TRM_F6
      EXTERNAL LONG CONSTANT SMG$K_TRM_F20
      EXTERNAL LONG CONSTANT SMG$K_TRM_FIND
      EXTERNAL LONG CONSTANT SMG$K_TRM_NEXT_SCREEN
      EXTERNAL LONG CONSTANT SMG$K_TRM_TIMEOUT
      EXTERNAL LONG FUNCTION SMG$CREATE_VIRTUAL_KEYBOARD( LONG, STRING )
      EXTERNAL LONG FUNCTION SMG$DELETE_VIRTUAL_KEYBOARD( LONG )
      EXTERNAL LONG FUNCTION SMG$READ_KEYSTROKE( LONG, LONG, STRING, &
          LONG, LONG )

      !+
      ! Prompt the user for the timer value. A value of 0 will cause
      ! the type-ahead buffer to be read.
      !-

      INPUT "Enter timer value (0 to read type-ahead buffer): ";TIMER

      !+
      ! Establish a SMG connection to SYS$INPUT. Signal any unexpected
      ! errors.
      !-

      RET_STATUS = SMG$CREATE_VIRTUAL_KEYBOARD( KB_ID, "SYS$INPUT:" )
      IF (RET_STATUS AND 1%) = 0% THEN
          CALL LIB$SIGNAL( RET_STATUS )
      END IF
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-2 (Cont.) Using SMG\$ Routines in VAX BASIC

```
!+
!   Read a keystroke, tell the user what we found.
!-

RET_STATUS = SMG$READ_KEYSTROKE( KB_ID, TERM_CODE, , TIMER, )
IF (RET_STATUS <> SS$TIMEOUT) AND ((RET_STATUS AND 1%) = 0%) THEN
    CALL LIB$SIGNAL( RET_STATUS )
END IF

PRINT "term_code = ";TERM_CODE

SELECT TERM_CODE
    CASE 0 TO 31
        PRINT "You typed a control character"
    CASE 32 TO 127
        PRINT "You typed: ";CHR$(TERM_CODE)
    CASE SMG$K_TRM_PF1 TO SMG$K_TRM_PERIOD
        PRINT "You typed one of the keypad keys"
    CASE SMG$K_TRM_UP TO SMG$K_TRM_RIGHT
        PRINT "You typed one of the cursor positioning keys"
    CASE SMG$K_TRM_F6 TO SMG$K_TRM_F20
        PRINT "You typed one of the function keys"
    CASE SMG$K_TRM_FIND TO SMG$K_TRM_NEXT_SCREEN
        PRINT "You typed one of the editing keys"
    CASE SMG$K_TRM_TIMEOUT
        PRINT "You did not type a key fast enough"
    CASE ELSE
        PRINT "I'm not sure what key you typed"
END SELECT

!+
!   Close the connection to SYS$INPUT, and signal any errors.
!-

RET_STATUS = SMG$DELETE_VIRTUAL_KEYBOARD( KB_ID )
IF (RET_STATUS AND 1%) = 0% THEN
    CALL LIB$SIGNAL( RET_STATUS )
END IF

END
```

Examples of Calling SMG\$ Routines

The BLISS program shown in Example 7-3 demonstrates the use of SMG\$READ_KEYSTROKE from a lower-level language.

Example 7-3 Using SMG\$ Routines in VAX BLISS32

```
MODULE READ_SINGLE_CHAR (      MAIN = PERFORM_READ,
                               %TITLE 'Read a Keystroke from SYS$INPUT'
                               IDENT = '1-001' ) =
BEGIN
!+
! Facility:      Example programs
!
! Abstract:      This example program uses the routine SMG$READ_KEYSTROKE
!                to get a single character input from the current SYS$INPUT
!                device and then indicates the nature of the input to the user.
!
! Environment:   User mode, AST reentrant
!
! Author:        John Doe          Creation Date: 8-Apr-1985
!
! Modified by:
! 1-001 - Original.  JD 8-Apr-1985
!-

!+
! General mode addressing must be used for external references.
!-

%SBTTL 'Declarations'
SWITCHES ADDRESSING_MODE (EXTERNAL=GENERAL, NONEXTERNAL=WORD_RELATIVE);

!+
! Obtain SMG$, SS$, etc. definitions.
!-

LIBRARY 'SYS$LIBRARY:STARLET';

!+
! Use the TUTIO package for the purposes of this small example.
!-

REQUIRE 'SYS$LIBRARY:TUTIO';

!+
! Declare screen management routines used by this program, as well as
! any other external routines.
!-

EXTERNAL ROUTINE
    SMG$CREATE_VIRTUAL_KEYBOARD,
    SMG$DELETE_VIRTUAL_KEYBOARD,
    SMG$READ_KEYSTROKE,
    LIB$SIGNAL : NOVALUE;

!+
! Define a convenient way to check the return status from a routine.
!-
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-3 (Cont.) Using SMG\$ Routines in VAX BLISS32

```
MACRO CHECK (X) =
IF NOT X
THEN
    LIB$SIGNAL (X)
%;

%SBTTL 'Routine PERFORM_READ'
ROUTINE PERFORM_READ: NOVALUE =

!+
! Functional Description:
!
!     This routine uses screen management I/O to get a single character
!     input from the current SYS$INPUT device, and then processes it by
!     what its character or termination code is.
!
! Calling Sequence:
!
!     Not Callable
!
! Formal Arguments:
!
!     Not Applicable
!
! Implicit Inputs:
!
!     None
!
! Implicit Outputs:
!
!     None
!
! Side Effects:
!
!     Any error returned by screen management routines except for
!     SSS_TIMEOUT will be signaled.
!-

BEGIN

    LITERAL
        ZERO = 0;

    LOCAL
        KBID : INITIAL(0),
        TERM_CODE : INITIAL(0),
        TIMER_VALUE : INITIAL(0),
        SMG_STATUS;

!+
! Obtain a read timeout value.
!-
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-3 (Cont.) Using SMG\$ Routines in VAX BLISS32

```
TIMER_VALUE = 10;

!+
! Establish a screen managed connection to SYS$INPUT.
!-

SMG_STATUS = SMG$CREATE_VIRTUAL_KEYBOARD (KBID, %ASCID'SYS$INPUT');
CHECK (.SMG_STATUS);

!+
! Read a keystroke and tell the user what was found.
!-

SMG_STATUS = SMG$READ_KEYSTROKE (KBID, TERM_CODE, ZERO, TIMER_VALUE);
IF (.SMG_STATUS NEQ SS$TIMEOUT)
THEN
    CHECK (.SMG_STATUS);

SELECTONE .TERM_CODE OF
SET
    [0 TO 31]:
    TTY_PUT_QUO ('You typed a control character.');
```

```
    [32 TO 127]:
    TTY_PUT_QUO ('You typed a printable character.');
```

```
    [SMG$K_TRM_PF1 TO SMG$K_TRM_PERIOD]:
    TTY_PUT_QUO ('You typed one of the keypad keys.');
```

```
    [SMG$K_TRM_UP TO SMG$K_TRM_RIGHT]:
    TTY_PUT_QUO ('You typed one of the cursor positioning keys.');
```

```
    [SMG$K_TRM_F6 TO SMG$K_TRM_F20]:
    TTY_PUT_QUO ('You typed one of the function keys.');
```

```
    [SMG$K_TRM_FIND TO SMG$K_TRM_NEXT_SCREEN]:
    TTY_PUT_QUO ('You typed one of the editing keys.');
```

```
    [SMG$K_TRM_TIMEOUT]:
    TTY_PUT_QUO ('You did not type a key fast enough.');
```

```
    [OTHERWISE]:
    TTY_PUT_QUO ('I am not sure what you typed.');
```

```
TES;
TTY_PUT_CRLF ();
!+
! Terminate the screen managed connection to SYS$INPUT.
!-

SMG_STATUS = SMG$DELETE_VIRTUAL_KEYBOARD (KBID);
CHECK (.SMG_STATUS);

END;

END
ELUDOM
```


Examples of Calling SMG\$ Routines

Example 7-4 shows the techniques used to call SMG\$READ_KEYSTROKE from VAX COBOL.

Example 7-4 Using SMG\$ Routines in VAX COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      KEYSTROKE.
*
* This routine creates a VIRTUAL DISPLAY and writes it to the PASTEBOARD.
* Data is placed in the VIRTUAL DISPLAY using the routine SMG$PUT_LINE.
* SMG$READ_KEYSTROKE is called to read a keystroke from the VIRTUAL KEYBOARD.
*
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DISPLAY1          PIC 9(9)  COMP.
01 PASTE1            PIC 9(9)  COMP.
01 KEYBOARD1        PIC 9(9)  COMP.
01 ROWS              PIC S9(9) COMP    VALUE 7.
01 COLUMNS          PIC S9(9) COMP    VALUE 60.
01 DISPLAY_NAME     PIC X(13) VALUE " DISPLAY ONE ".
01 TERM_CHAR        PIC 9(4)  COMP.
01 T_TEXT           PIC X(6).
01 TEXT_OUTPUT      PIC X(24) VALUE " TERMINAL CHARACTER IS: ".
01 PROMPT           PIC X(2)  VALUE ">>".
01 LINE_1           PIC X(12) VALUE "Hit any key.".
01 LINE_2           PIC X(34) VALUE "This character will not be echoed.".
01 LINE_3           PIC X(47) VALUE "The terminal character equivalent is displayed.".
01 LINE_4           PIC X      VALUE " ".
01 THREE            PIC S9(9) COMP    VALUE 3.
01 NINE             PIC S9(9) COMP    VALUE 9.
01 SEVEN            PIC S9(9) COMP    VALUE 7.
01 TWENTY_FIVE     PIC S9(9) COMP    VALUE 25.
01 CHAR_SET         PIC S9(9) COMP    VALUE EXTERNAL SMG$C_HANYU.
PROCEDURE DIVISION.
P0.
* Create the virtual display with a border.
    CALL "SMG$CREATE_VIRTUAL_DISPLAY" USING
        ROWS, COLUMNS, DISPLAY1,
        OMITTED, OMITTED, CHAR_SET.
* Create the pasteboard
    CALL "SMG$CREATE_PASTEBOARD" USING PASTE1.
* Create a virtual keyboard
    CALL "SMG$CREATE_VIRTUAL_KEYBOARD" USING KEYBOARD1.
* Paste the virtual display at row 3, column 9.
    CALL "SMG$LABEL_BORDER" USING DISPLAY1, BY DESCRIPTOR DISPLAY_NAME,
        OMITTED, OMITTED, OMITTED, OMITTED, CHAR_SET.
    CALL "SMG$PASTE_VIRTUAL_DISPLAY" USING
        DISPLAY1, PASTE1, THREE, NINE.
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-4 (Cont.) Using SMG\$ Routines in VAX COBOL

* Place data in the virtual display

```
CALL "SMG$PUT_LINE" USING DISPLAY1, BY DESCRIPTOR LINE_1,  
    OMITTED, OMITTED, OMITTED, OMITTED, CHAR_SET.  
CALL "SMG$PUT_LINE" USING DISPLAY1, BY DESCRIPTOR LINE_2,  
    OMITTED, OMITTED, OMITTED, OMITTED, CHAR_SET.  
CALL "SMG$PUT_LINE" USING DISPLAY1, BY DESCRIPTOR LINE_3,  
    OMITTED, OMITTED, OMITTED, OMITTED, CHAR_SET.  
CALL "SMG$PUT_LINE" USING DISPLAY1, BY DESCRIPTOR LINE_4.
```

* Read a keystroke from the virtual pasteboard.

```
CALL "SMG$READ_KEYSTROKE" USING KEYBOARD1, TERM_CHAR,  
    BY DESCRIPTOR PROMPT, OMITTED, BY REFERENCE DISPLAY1.  
CALL "SMG$PUT_LINE" USING DISPLAY1, BY DESCRIPTOR LINE_4.
```

* Convert the decimal value of TERM_CHAR to a decimal ASCII text string.

```
CALL "OTS$CVT_L_TI" USING TERM_CHAR, BY DESCRIPTOR T_TEXT.
```

* Print out the decimal ASCII text string.

```
CALL "SMG$PUT_LINE" USING DISPLAY1, BY DESCRIPTOR TEXT_OUTPUT.  
CALL "SMG$PUT_CHARS" USING DISPLAY1, BY DESCRIPTOR T_TEXT,  
    BY REFERENCE SEVEN, TWENTY_FIVE.
```

```
STOP RUN.
```

Examples of Calling SMG\$ Routines

The FORTRAN program shown in Example 7-5 uses SMG\$READ_KEYSTROKE, as well as SMG\$CREATE_VIRTUAL_DISPLAY, SMG\$CREATE_PASTEBOARD, SMG\$PASTE_VIRTUAL_DISPLAY, SMG\$CREATE_VIRTUAL_KEYBOARD, and SMG\$PUT_LINE.

Example 7-5 Using SMG\$ Routines in DEC Fortran

```
C+
C This routine creates a virtual display and writes it to the PASTEBOARD.
C Data is placed in the virtual display using the routine SMG$PUT_CHARS.
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-
      INCLUDE '($SMGDEF)'
      INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
      INTEGER SMG$PASTE_VIRTUAL_DISPLAY,
1         SMG$CREATE_VIRTUAL_KEYBOARD
      INTEGER SMG$READ_KEYSTROKE, SMG$PUT_LINE
      INTEGER DISPLAY1, PASTE1, KEYBOARD1, ROWS, COLUMNS,
1         TERM_CHAR
      CHARACTER*3 TEXT
      CHARACTER*27 TEXT_OUTPUT

C+
C Create the virtual display with a border.
C-
      ROWS = 7
      COLUMNS = 60

      ISTATUS = SMG$CREATE_VIRTUAL_DISPLAY
1         (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER, , SMG$C_HANYU)

C+
C Create the pasteboard.
C-
      ISTATUS = SMG$CREATE_PASTEBOARD (PASTE1)

C+
C Create a virtual keyboard.
C-
      ISTATUS = SMG$CREATE_VIRTUAL_KEYBOARD ( KEYBOARD1)

C+
C Paste the virtual display at row 3, column 9.
C-
      ISTATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 3, 9)
      ISTATUS = SMG$PUT_LINE (DISPLAY1,
1         'Enter the character K after the >> prompt.',
1         , , , SMG$C_HANYU )
      ISTATUS = SMG$PUT_LINE (DISPLAY1,
1         'This character will not be echoed as you type it.',
1         , , , SMG$C_HANYU )
      ISTATUS = SMG$PUT_LINE (DISPLAY1,
1         'The terminal character equivalent of K is displayed.',
1         , , , SMG$C_HANYU )
      ISTATUS = SMG$PUT_LINE (DISPLAY1, ' ')

C+
C Read a keystroke from the virtual pasteboard.
C-
      ISTATUS = SMG$READ_KEYSTROKE ( KEYBOARD1, TERM_CHAR, '>>', ,
1         DISPLAY1)

      ISTATUS = SMG$PUT_LINE (DISPLAY1, ' ')
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-5 (Cont.) Using SMG\$ Routines in DEC Fortran

```
C+
C Convert the decimal value of TERM_CHAR to a decimal ASCII text string.
C-
      ISTATUS = OTS$CVT_L_TI( TERM_CHAR, TEXT)
      TEXT_OUTPUT = ' TERMINAL CHARACTER IS: ' // TEXT
C+
C Print the decimal ASCII text string.
C-
      ISTATUS = SMG$PUT_LINE (DISPLAY1, TEXT_OUTPUT, , , , , SMG$C_HANYU)
      ISTATUS = SMG$PUT_CHARS (DISPLAY1, TEXT, 7, 25, , , , SMG$C_HANYU)
      END
```

Examples of Calling SMG\$ Routines

The VAX MACRO program shown in Example 7–6 demonstrates the precise steps required to call SMG\$READ_KEYSTROKE from a low-level language.

Example 7–6 Using SMG\$ Routines in VAX MACRO

```
.TITLE  SMG_DEMO
;+
; This program demonstrates the use of the SMG$ routines, in particular
; SMG$READ_KEYSTROKE.
;-
        $DSCDEF          ; Declare DSC$ symbols
        $SMGDEF          ; Declare SMG$ symbols
;+
; Declare external routines.
;-
        .EXTRN  SMG$CREATE_PASTEBOARD
        .EXTRN  SMG$CREATE_VIRTUAL_DISPLAY
        .EXTRN  SMG$CREATE_VIRTUAL_KEYBOARD
        .EXTRN  SMG$PUT_LINE
        .EXTRN  SMG$READ_KEYSTROKE
;+
; Declare data PSECT and objects.
;-
        .PSECT  $DATA RD,WRT,NOEXE,NOSHR,PIC
LINE1:  .ASCID  "Enter the character K after the prompt."
LINE2:  .ASCID  "This character will not be echoed as you type it."
LINE3:  .ASCID  "The terminal character equivalent of K is displayed."
PROMPT: .ASCID  ">>"
BLANK:  .ASCID  " "
FAOSTR: .ASCID  "TERMINAL CHARACTER IS !UL"
TEXT:   .BLKB   80      ; Buffer for formatted text
TEXT_LEN = . - TEXT    ; Length of TEXT
TEXT_DSC:          ; Descriptor for TEXT string
        .WORD   TEXT_LEN      ; DSC$W_LENGTH
        .BYTE   DSC$K_DTYPE_T ; DSC$B_DTYPE
        .BYTE   DSC$K_CLASS_S ; DSC$B_CLASS
        .ADDRESS TEXT         ; DSC$A_POINTER
CHAR_SET:
        .LONG   SMG$C_HANYU    ; Character set value
TERM_CHAR:
        .BLKL          ; Space for terminator character code
PASTEBOARD_1:
        .BLKL          ; Pasteboard ID
DISPLAY_1:
        .BLKL          ; Display ID
KEYBOARD_1:
        .BLKL          ; Keyboard ID
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-6 (Cont.) Using SMG\$ Routines in VAX MACRO

```
;+
; Declare PSECT for code.
;-
        .PSECT  $CODE RD,NOWRT,EXE,SHR,PIC
;+
; Begin main routine.
;-
        .ENTRY  SMG_DEMO, ^M<> ; Save no registers
;+
; Create virtual display.
;-
        PUSHL  #SMG$M_BORDER    ; Put flag on stack
        PUSHL  #60              ; Put columns on stack
        PUSHL  #7               ; Put rows on stack
        PUSHAB L^CHAR_SET      ; Address of character set
        PUSHL  #0               ; Do not specify video attribute
        PUSHAB 16(SP)          ; Address of flag
        PUSHAB L^DISPLAY_1     ; Address of display ID
        PUSHAB 20(SP)          ; Address of columns
        PUSHAB 20(SP)          ; Address of rows
        CALLS  #6, G^SMG$CREATE_VIRTUAL_DISPLAY
        ADDL2  #12, SP         ; Pop off temporaries

; Create pasteboard.
        PUSHAB L^PASTEBOARD_1  ; Address of pasteboard
        CALLS  #1, G^SMG$CREATE_PASTEBOARD

; Create virtual keyboard.
        PUSHAB L^KEYBOARD_1    ; Address of keyboard
        CALLS  #1, G^SMG$CREATE_VIRTUAL_KEYBOARD

; Paste the virtual display at row 3, column 9.
        PUSHL  #9              ; Put column on stack
        PUSHL  #3              ; Put row on stack
        PUSHAB 4(SP)           ; Address of column
        PUSHAB 4(SP)           ; Address of row
        PUSHAB L^PASTEBOARD_1  ; Address of pasteboard
        PUSHAB L^DISPLAY_1     ; Address of display
        CALLS  #4, G^SMG$PASTE_VIRTUAL_DISPLAY
        ADDL2  #8, SP         ; Pop off temporaries
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-6 (Cont.) Using SMG\$ Routines in VAX MACRO

```
; Write instructions.
    PUSHAB L^CHAR_SET      ; Character set
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHAB L^LINE1        ; "Enter the character..."
    PUSHAB L^DISPLAY_1    ; Display ID
    CALLS  #7, G^SMG$PUT_LINE
    PUSHAB L^CHAR_SET      ; Character set
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHAB L^LINE2        ; "This character will not..."
    PUSHAB L^DISPLAY_1    ; Display ID
    CALLS  #7, G^SMG$PUT_LINE
    PUSHAB L^CHAR_SET      ; Character set
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHAB L^LINE3        ; "The terminal character..."
    PUSHAB L^DISPLAY_1    ; Display ID
    CALLS  #7, G^SMG$PUT_LINE
    PUSHAB L^BLANK        ; Blank line
    PUSHAB L^DISPLAY_1    ; Display ID
    CALLS  #2, G^SMG$PUT_LINE

; Read a keystroke from the virtual keyboard.
    PUSHAB L^DISPLAY_1    ; Display ID
    CLRL   -(SP)          ; No timeout
    PUSHAB L^PROMPT       ; Prompt string
    PUSHAB L^TERM_CHAR    ; Longword for terminator code
    PUSHAB L^KEYBOARD_1   ; Keyboard ID
    CALLS  #5, G^SMG$READ_KEYSTROKE

; Format the terminator code using $FAO.
    $FAO_S CTRSTR=L^FAOSTR,-                ; FAO control string
           OUTLEN=L^TEXT_DSC+DSC$W_LENGTH,- ; Output string length
           OUTBUF=L^TEXT_DSC,-             ; Output buffer
           P1=L^TERM_CHAR                  ; Value to format

; Display the formatted text.
    PUSHAB L^BLANK        ; Blank line
    PUSHAB L^DISPLAY_1    ; Display ID
    CALLS  #2, G^SMG$PUT_LINE
    PUSHAB L^CHAR_SET      ; Character set
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHL  #0              ; Neglect optional parameters
    PUSHAB L^TEXT_DSC      ; Text to display
    PUSHAB L^DISPLAY_1    ; Display ID
    CALLS  #7, G^SMG$PUT_LINE

; Return with status from last call.
    RET

.END    SMG_DEMO          ; Specify SMG_DEMO as main program
```

Examples of Calling SMG\$ Routines

Example 7-7 uses SMG\$READ_KEYSTROKE from VAX Pascal. It also demonstrates the use of SMG\$CREATE_VIRTUAL_DISPLAY, SMG\$CREATE_PASTEBOARD, SMG\$CREATE_VIRTUAL_KEYBOARD, SMG\$PASTE_VIRTUAL_DISPLAY, and SMG\$PUT_LINE.

Example 7-7 Using SMG\$ Routines in VAX Pascal

```
{ This program demonstrates the use of the SMG$ routines, in particular }
{ SMG$READ_KEYSTROKE. }

[INHERIT('SYS$LIBRARY:STARLET')]
PROGRAM SMG_DEMO;

TYPE
    UNSIGNED_WORD = [WORD] 0..65535;

FUNCTION SMG$CREATE_VIRTUAL_DISPLAY (
    ROWS, COLUMNS: INTEGER;
    VAR DISPLAY_ID: INTEGER;
    DISPLAY_ATTRIBUTES, VIDEO_ATTRIBUTES, CHAR_SET: UNSIGNED
        := %IMMED 0): UNSIGNED; EXTERN;

FUNCTION SMG$CREATE_PASTEBOARD (
    VAR PASTEBOARD_ID: INTEGER;
    OUTPUT_DEVICE: PACKED ARRAY [A..B:INTEGER] OF CHAR := %IMMED 0;
    ROWS, COLUMNS: INTEGER := %IMMED 0;
    PRESERVE_SCREEN_FLAG: BOOLEAN := %IMMED 0) : UNSIGNED; EXTERN;

FUNCTION SMG$CREATE_VIRTUAL_KEYBOARD (
    VAR KEYBOARD_ID: INTEGER;
    FILESPEC: PACKED ARRAY [A..B:INTEGER] OF CHAR := %IMMED 0;
    DEFAULT_FILESPEC: PACKED ARRAY [C..D:INTEGER] OF CHAR := %IMMED 0;
    RESULTANT_FILESPEC: PACKED ARRAY [E..F:INTEGER] OF CHAR := %IMMED 0
    ): UNSIGNED; EXTERN;

FUNCTION SMG$PASTE_VIRTUAL_DISPLAY (
    DISPLAY_ID, PASTEBOARD_ID: INTEGER;
    ROW, COLUMN: INTEGER): UNSIGNED; EXTERN;

FUNCTION SMG$READ_KEYSTROKE (
    KEYBOARD_ID: INTEGER;
    VAR TERMINATOR_CODE: UNSIGNED_WORD;
    PROMPT: PACKED ARRAY [A..B:INTEGER] OF CHAR := %IMMED 0;
    TIMEOUT, DISPLAY_ID: INTEGER := %IMMED 0): UNSIGNED; EXTERN;

FUNCTION SMG$PUT_LINE (
    DISPLAY_ID: INTEGER;
    TEXT: PACKED ARRAY [A..B:INTEGER] OF CHAR;
    LINE_ADVANCE: INTEGER := %IMMED 0;
    RENDITION_SET, RENDITION_COMPLEMENT: UNSIGNED := %IMMED 0;
    WRAP_FLAG: BOOLEAN := %IMMED 0;
    CHAR_SET: UNSIGNED := %IMMED 0): UNSIGNED; EXTERN;
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-7 (Cont.) Using SMG\$ Routines in VAX Pascal

```
var
    PASTEBOARD_1, DISPLAY_1, KEYBOARD_1: INTEGER;
    TERMINATOR: UNSIGNED_WORD;

BEGIN
    { Create virtual display, pasteboard and virtual keyboard }
    SMG$CREATE_VIRTUAL_DISPLAY (ROWS := 7, COLUMNS := 60,
        DISPLAY_ID := DISPLAY_1,
        DISPLAY_ATTRIBUTES := SMG$M_BORDER,
        CHAR_SET := SMG$C_HANYU);
    SMG$CREATE_PASTEBOARD (PASTEBOARD_ID := PASTEBOARD_1);
    SMG$CREATE_VIRTUAL_KEYBOARD (KEYBOARD_ID := KEYBOARD_1);

    { Paste the virtual display at row 3, column 9 }
    SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY_ID := DISPLAY_1,
        PASTEBOARD_ID := PASTEBOARD_1, ROW := 3, COLUMN := 9);

    { Write the instructions to the virtual display }
    SMG$PUT_LINE (DISPLAY_ID := DISPLAY_1,
        TEXT := 'Enter the character K after the >> prompt.',
        CHAR_SET := SMG$C_HANYU);
    SMG$PUT_LINE (DISPLAY_ID := DISPLAY_1,
        TEXT := 'This character will not be echoed as you type it.',
        CHAR_SET := SMG$C_HANYU);
    SMG$PUT_LINE (DISPLAY_ID := DISPLAY_1,
        TEXT := 'The terminal character equivalent of K is displayed.',
        CHAR_SET := SMG$C_HANYU);
    SMG$PUT_LINE (DISPLAY_ID := DISPLAY_1,
        TEXT := ' ',
        CHAR_SET := SMG$C_HANYU);

    { Read the keystroke from the virtual keyboard }
    SMG$READ_KEYSTROKE (KEYBOARD_ID := KEYBOARD_1,
        DISPLAY_ID := DISPLAY_1,
        TERMINATOR_CODE := TERMINATOR, PROMPT := '>>');

    { Display the decimal value of the terminator code }
    SMG$PUT_LINE (DISPLAY_ID := DISPLAY_1,
        TEXT := ' ',
        CHAR_SET := SMG$C_HANYU);
    SMG$PUT_LINE (DISPLAY_ID := DISPLAY_1,
        TEXT := 'TERMINAL CHARACTER IS ' + DEC(TERMINATOR,5,1),
        CHAR_SET := SMG$C_HANYU);

END.
```

Examples of Calling SMG\$ Routines

The program shown in Example 7–8 calls SMG\$READ_KEYSTROKE from VAX PL/I.

Example 7–8 Using SMG\$ Routines in VAX PL/I

```
/*
 * Example of SMG$READ_KEYSTROKE.
 */

/*
 * Declare the RTL entry points.
 */
declare
    SMG$CREATE_VIRTUAL_KEYBOARD external entry(
        fixed binary(31),      /* new-keyboard-id */
        character(*),         /* filespec */
        character(*),         /* default-filespec */
        character(*) varying ) /* resultant-filespec */
    returns(fixed binary(31)) options(variable);

declare
    SMG$DELETE_VIRTUAL_KEYBOARD external entry(
        fixed binary(31) )     /* keyboard-id */
    returns(fixed binary(31));

declare
    SMG$READ_KEYSTROKE external entry(
        fixed binary(31),     /* keyboard-id */
        fixed binary(15),    /* terminator-code */
        character(*),        /* prompt-string */
        fixed binary(31),    /* timeout */
        fixed binary(31) )   /* display-id */
    returns(fixed binary(31)) options(variable);

/*
 * Get the value of the SMG constants from PLISTARLET.
 */
#include $SMGDEF;
declare SMG$_EOF globalref value fixed binary(31);

/*
 * Misc. constants.
 */
%replace false by '0'b;
%replace true by '1'b;

/*
 * The following compile-time routine will signal an error at run-time
 * if the status value that it is passed does not have success or
 * informational severity. (i.e. if the low bit is not set.)
 */
%signal_if: procedure (status_val) returns(character);
    %declare status_val character;
    %return( 'if posint(' || status_val || ',1,1) = 0 ' ||
            'then signal vaxcondition(' || status_val || ')' );
%end;
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-8 (Cont.) Using SMG\$ Routines in VAX PL/I

```
main: proc options(main, ident('V4.2'));

    declare exit bit initial(false);
    declare status fixed binary(31);
    declare keyboard_id fixed binary(31);
    declare terminator fixed binary(15);

    /*
     * Create the virtual keyboard necessary for the read.
     */
    status = smg$create_virtual_keyboard( keyboard_id );
    signal_if( status );
    /*
     * Read a single keystroke. If that keystroke is an end-of-file,
     * then exit. Otherwise, SELECT the appropriate action based on
     * the key.
     */
    do while(^exit);

        status = smg$read_keystroke( keyboard_id, terminator,
            'Command: ', 20 );

        if status = SMG$_EOF
            then exit = true;

        else do;
            signal_if( status );
            select (terminator);

                when (SMG$K_TRM_PF2,
                    SMG$K_TRM_HELP,
                    rank('H'),
                    rank('h'),
                    rank('?') )    call display_help;

                when(SMG$K_TRM_DO) call do_command;

                when(rank('E'),
                    rank('e'))    exit = true;

                otherwise          call command_error;

            end;
        end;

    end;

    /*
     * We're done, so delete the virtual keyboard.
     */
    status = smg$delete_virtual_keyboard( keyboard_id );
    signal_if( status );

end main;
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-8 (Cont.) Using SMG\$ Routines in VAX PL/I

```
display_help: procedure;
    put skip edit('This program uses single keystroke commands.') (A);
    put skip edit('The following keys are valid:') (A);
    put skip;
    put skip edit('      Key           Function') (A);
    put skip edit('      E/e           Exit') (A);
    put skip edit('      <DO> Your choice...') (A);
    put skip edit('      ?/H/h/<HELP> Help') (A);
    put skip;
    end display_help;

do_command: procedure;
    put skip edit('The DO key was pressed') (A);
    put skip;
    end do_command;

command_error: procedure;
    put skip edit('The key pressed was not valid - please try again.') (A);
    put skip edit('(H for HELP).') (A);
    put skip;
    end command_error;
```

Examples of Calling SMG\$ Routines

Example 7–9 demonstrates how to call SMG\$READ_KEYSTROKE from VAX RPG II. This program also uses SMG\$CREATE_VIRTUAL_KEYBOARD and SMG\$DELETE_VIRTUAL_KEYBOARD.

This RPG II program displays the following if the cursor positioning and control keys are typed:

UP DOWN RIGHT LEFT

These keys include the arrow keys (up, down, right, and left) and Ctrl/Z.

Example 7–9 Using SMG\$ Routines in VAX RPG II

```

0      | 1      | 2      | 3      | 4      | 5      | 6      |
1234567890123456789012345678901234567890123456789012345678901234567890

F*+
F* This RPG II program demonstrates the use of the RTL routine
F* SMG$READ_KEYSTROKE to read a keystroke from the terminal.
F*
F* The program takes input from the terminal until Ctrl/Z is
F* typed. If any of the four cursor positioning keys is typed,
F* a string is displayed corresponding to the key.
F*
F* Build this program using the following commands:
F*
F* $ RPG READ_KEY
F* $ CREATE SMGDEF.MAR
F*      .TITLE SMGDEF - Define SMG$ constants
F*      .Ident /1-000/
F*
F*      $SMGDEF GLOBAL
F*      .END
F* $ MACRO SMGDEF
F* $ LINK READ_KEY,SMGDEF
F*-
FTTY   D   V       5           TTY
C* External definitions for SMG routines.
C      CREKB      EXTRN' SMG$CREATE_VIRTUAL_KEYBOARD'
C      DELKB      EXTRN' SMG$DELETE_VIRTUAL_KEYBOARD'
C      REAKEY      EXTRN' SMG$READ_KEYSTROKE'
C* External definitions for SMG terminators.
C      T_UP       EXTRN' SMG$K_TRM_UP'
C      T_DOWN     EXTRN' SMG$K_TRM_DOWN'
C      T_LEFT     EXTRN' SMG$K_TRM_LEFT'
C      T_RIGHT    EXTRN' SMG$K_TRM_RIGHT'
C      T_CTRLZ    EXTRN' SMG$K_TRM_CTRLZ'
C* Create the virtual keyboard.
C N99          CALL CREKB
C              PARM          KB_ID  90 WL
C              SETON          99
```

(continued on next page)

Examples of Calling SMG\$ Routines

Example 7-9 (Cont.) Using SMG\$ Routines in VAX RPG II

```
C* Read a keystroke.
C          CALL REAKEY
C          PARM          KB_ID  90 RL
C          PARM          T_CODE 50 WW
C* Turn on an indicator if a cursor positioning key was typed.
C          T_CODE COMP T_UP          01
C          T_CODE COMP T_DOWN        02
C          T_CODE COMP T_LEFT        03
C          T_CODE COMP T_RIGHT       04
C* Turn on LR to quit if Ctrl/Z was typed.
C          T_CODE COMP T_CTRLZ      LR
C* Display a message if a cursor positioning key was typed.
C  01      'UP'      DSPLYTTY
C  02      'DOWN'    DSPLYTTY
C  03      'LEFT'    DSPLYTTY
C  04      'RIGHT'   DSPLYTTY
C* Delete the virtual keyboard.
CLR       CALL DELKB
CLR       PARM          KB_ID  90 RL
```

SMG\$ Reference Section

This section contains detailed descriptions of all routines provided by the Chinese RTL Screen Management (SMG\$) Facility.

SMG\$ADD_KEY_DEF—Add Key Definition

The Add Key Definition routine adds a keypad key definition to a table of key definitions.

Format

```
SMG$ADD_KEY_DEF key-table-id ,key-name [,if-state] [,attributes] [,equivalence-string] [,state-string]
```

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

key-table-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Identifies the key table to which you are adding a key definition. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

Key-table-id is returned by the SMG\$CREATE_KEY_TABLE routine.

key-name

OpenVMS usage	char_string
type	character string
access	read only
mechanism	by descriptor

Identifies the key whose value you are defining. The **key-name** argument is the address of a descriptor pointing to this key name. The SMG\$ADD_KEY_DEF routine changes the string to uppercase and removes trailing blanks.

Table 3–1 in Chapter 3 lists the valid key names.

if-state

OpenVMS usage	char_string
type	character string
access	read only
mechanism	by descriptor

Qualifies the value returned when **key-name** is struck. The **if-state** argument is the address of a descriptor pointing to the state string.

If **if-state** is specified, this definition of **key-name** is used only if the current state matches the specified **if-state** string. The **if-state** argument must be from 1 to 31 characters in length. If this argument is omitted, **if-state** defaults to the value "DEFAULT."

SMG\$ADD_KEY_DEF

attributes

OpenVMS usage	mask_longword
type	longword (unsigned)
access	read only
mechanism	by reference

Longword bit mask specifying additional attributes of this key definition. The **attributes** argument is the address of an unsigned longword that contains this attribute mask. If omitted, the mask is zero.

Valid **attributes** are described in the following list:

SMG\$M_KEY_NOECHO	If set, this bit specifies that equivalence-string is not to be echoed when this key is pressed. If clear, equivalence-string is echoed. If SMG\$M_KEY_TERMINATE is not set, SMG\$M_KEY_NOECHO is ignored.
SMG\$M_KEY_TERMINATE	If set, this bit specifies that when this key is pressed (as qualified by if-state) the input line is complete and more characters should not be accepted. If clear, more characters may be accepted. In other words, setting this bit causes equivalence-string to be treated as a terminator.
SMG\$M_KEY_LOCK	If set, and if state-string is specified, the state name specified by state-string remains the current state until explicitly changed by a subsequent keystroke whose definition includes a state-string . If clear, the state name specified by state-string remains in effect only for the next defined keystroke.
SMG\$M_KEY_PROTECTED	If set, this bit specifies that this key definition cannot be modified or deleted. If clear, the key definition can be modified or deleted.

The remaining bits are undefined and must be zero. It is possible to perform a logical OR operation on these values to set more than one attribute at a time.

equivalence-string

OpenVMS usage	char_string
type	character string
access	read only
mechanism	by descriptor

Character string to be substituted for the keystroke in the returned line. The **equivalence-string** argument is the address of a descriptor pointing to this equivalence string.

Equivalence-string is echoed unless SMG\$M_KEY_NOECHO is set. If **equivalence-string** is omitted, no equivalence string is defined for this key.

state-string

OpenVMS usage	char_string
type	character string
access	read only
mechanism	by descriptor

Contains a new state name which becomes the current state when this key is pressed. The **state-string** argument is the address of a descriptor pointing to the new state string.

If omitted, no new state is defined. If the current state is temporary (that is, if SMG\$M_KEY_LOCK was not specified for the most recently pressed defined key), the current **state-string** becomes DEFAULT.

Description

SMG\$ADD_KEY_DEF inserts a key definition into a key definition table. The table must have been created with a call to SMG\$CREATE_KEY_TABLE. After SMG\$ADD_KEY_DEF executes, the specified equivalence string is returned when the user types the specified key in response to the SMG\$READ_COMPOSED_LINE routine.

You can define all keys on the VT100, VT200-series, and VT300-series keyboards and keypads with the following exceptions:

- The Compose Character key on VT200-series and VT300-series keyboards
- The ESCAPE key
- The SHIFT keys
- The keys F1 through F5 on VT200-series and VT300-series keyboards

There are some keys and key definitions that you can define but that Digital strongly suggests you avoid defining. SMG\$ does not return an error when you use them as key names, but the definitions you assign to these key combinations are not executed unless you set your terminal in the following special ways at the DCL level.

- Ctrl/C, Ctrl/O, Ctrl/X, and F6 - To use a definition that you bind to these keys, you must first enter the DCL command SET TERMINAL/PASTHRU.
- Ctrl/T, Ctrl/Y - To use a definition that you bind to these keys, you must first enter either the DCL command SET TERMINAL/PASTHRU or SET NOCONTROL, or both.
- Ctrl/S, Ctrl/Q - To use a definition that you bind to these keys, you must first enter the DCL command SET TERMINAL/NOTTSYNC.

Digital does not recommend that you use these special terminal settings. The settings may cause unpredictable results if you do not understand all the implications of changing the default settings for giving the terminal driver control.

SMG\$ADD_KEY_DEF

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_PREDEFREP	Successful completion. The previous key definition has been replaced.
SMG\$_INVDEFATT	Invalid key definition attributes.
SMG\$_INVKEYNAM	Invalid key-name .
SMG\$_INVKTB_ID	Invalid key-table-id .
SMG\$_KEYDEFPRO	Key definition is protected against change or deletion.
SMG\$_WRONUMARG	Wrong number of arguments.

Any condition values returned by LIB\$COPY_DXDX.

SMG\$BEGIN_DISPLAY_UPDATE—Begin Batching of Display Updates

The Begin Batching of Display Updates routine saves, or batches, all output to a virtual display until a matching call to SMG\$END_DISPLAY_UPDATE is encountered.

Format

SMG\$BEGIN_DISPLAY_UPDATE display-id

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

display-id	
OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the virtual display for which output is to be batched. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG\$CREATE_VIRTUAL_DISPLAY.

Description

SMG\$BEGIN_DISPLAY_UPDATE lets you make more than one change to a display and have the changes appear only after all changes are complete. Thus, the user sees the display change from its initial state to its final state, without seeing any of the intermediate states.

Batching terminates when SMG\$END_DISPLAY_UPDATE has been called the same number of times for a given display as has SMG\$BEGIN_DISPLAY_UPDATE. The Chinese Screen Management Facility keeps track of batching for a given display; thus, the calls to the SMG\$BEGIN_DISPLAY_UPDATE and SMG\$END_DISPLAY_UPDATE need not occur in the same module.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_BATWAS_ON	Successful completion; batching has already been initiated.
SMG\$_WRONUMARG	Wrong number of arguments.
SMG\$_INVDIS_ID	Invalid display-id .

SMG\$BEGIN_PASTEBOARD_UPDATE

SMG\$BEGIN_PASTEBOARD_UPDATE—Begin Batching of Pasteboard Updates

The Begin Batching of Pasteboard Updates routine saves, or batches, all output to a pasteboard until a matching call to SMG\$END_PASTEBOARD_UPDATE is encountered.

Format

SMG\$BEGIN_PASTEBOARD_UPDATE pasteboard-id

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

pasteboard-id	
OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the pasteboard for which output is to be batched. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG\$CREATE_PASTEBOARD.

Description

SMG\$BEGIN_PASTEBOARD_UPDATE lets you make more than one change to a pasteboard and have the changes appear only after all changes are complete. Thus, the user sees the pasteboard change from its initial state to its final state, without seeing any of the intermediate states.

Batching terminates when SMG\$END_PASTEBOARD_UPDATE has been called the same number of times for a given pasteboard as has SMG\$BEGIN_PASTEBOARD_UPDATE. The Chinese Screen Management Facility keeps track of batching for a given pasteboard; thus, the calls to the SMG\$BEGIN_PASTEBOARD_UPDATE and SMG\$END_PASTEBOARD_UPDATE need not occur in the same module.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_BATWAS_ON	Successful completion; batching has already been initiated.
SMG\$_WRONUMARG	Wrong number of arguments.
SMG\$_INVPAS_ID	Invalid pasteboard-id .

SMG\$CANCEL_INPUT—Cancel Input Request

The Cancel Input Request routine immediately cancels any read-in-progress that was issued by SMG\$READ_COMPOSED_LINE, SMG\$READ_KEYSTROKE, or SMG\$READ_STRING.

Format

SMG\$CANCEL_INPUT keyboard-id

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

keyboard-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the virtual keyboard for which the input is to be canceled. The **keyboard-id** argument is the address of an unsigned longword that contains the keyboard identifier.

Keyboard-id is returned by SMG\$CREATE_VIRTUAL_KEYBOARD.

Description

SMG\$CANCEL_INPUT causes immediate termination of an SMG\$READ_COMPOSED_LINE, SMG\$READ_KEYSTROKE, or SMG\$READ_STRING input operation from a terminal. The condition code SS\$_CANCEL or SS\$_ABORT is returned to those routines when you use SMG\$CANCEL_INPUT.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_INVKBD_ID	Invalid keyboard-id .
SMG\$_WRONUMARG	Wrong number of arguments.

SMG\$CHANGE_PBD_CHARACTERISTICS

SMG\$CHANGE_PBD_CHARACTERISTICS—Change Pasteboard Characteristics

The Change Pasteboard Characteristics routine lets you change the characteristics associated with a pasteboard.

Format

```
SMG$CHANGE_PBD_CHARACTERISTICS  
                                pasteboard-id [,desired-width] [,width] [,desired-height] [,height]  
                                [,desired-background-color] [,background-color]
```

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

pasteboard-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the pasteboard whose characteristics are to be changed. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG\$CREATE_PASTEBOARD.

desired-width

OpenVMS usage	longword_signed
type	longword (signed)
access	read only
mechanism	by reference

New width for the pasteboard. The **desired-width** argument is the address of a signed longword that contains the desired width. If omitted, the width does not change.

width

OpenVMS usage	longword_signed
type	longword (signed)
access	write only
mechanism	by reference

Receives the physical width of the pasteboard. The **width** argument is the address of a signed longword into which is written the actual width of the pasteboard.

If the terminal cannot be set exactly to **desired-width**, **width** may be larger than **desired-width**. If the physical width of the terminal is smaller than **desired-width**, **width** may be smaller than **desired-width**.

SMG\$CHANGE_PBD_CHARACTERISTICS

desired-height

OpenVMS usage longword_signed
type longword (signed)
access read only
mechanism by reference

New height for the pasteboard. The **desired-height** argument is the address of a signed longword that contains the desired height of the pasteboard. If **desired-height** is omitted, the height does not change.

height

OpenVMS usage longword_signed
type longword (signed)
access write only
mechanism by reference

Receives the physical height of the pasteboard. The **height** argument is the address of a signed longword into which is written the actual height of the pasteboard.

If the terminal cannot be set exactly to **desired-height**, **height** may be larger than **desired-height**. If the physical height of the terminal is smaller than **desired-height**, **height** may be smaller than **desired-height**.

desired-background-color

OpenVMS usage longword_unsigned
type longword (unsigned)
access read only
mechanism by reference

Symbolic name for the desired background color. The **desired-background-color** argument is the address of an unsigned longword that contains the desired color.

The symbols listed below are defined in \$SMGDEF. Valid values for **desired-background-color** are as follows:

SMG\$C_COLOR_WHITE	White background
SMG\$C_COLOR_BLACK	Black background
SMG\$C_COLOR_BLUE	Blue background
SMG\$C_COLOR_CYAN	Cyan (green-blue) background
SMG\$C_COLOR_GREEN	Green background
SMG\$C_COLOR_MAGENTA	Magenta background
SMG\$C_COLOR_RED	Red background
SMG\$C_COLOR_YELLOW	Yellow background
SMG\$C_COLOR_LIGHT	Light background
SMG\$C_COLOR_DARK	Dark background
SMG\$C_COLOR_USER1	User-defined background 1
SMG\$C_COLOR_USER2	User-defined background 2

If you omit **desired-background-color**, or if the terminal hardware does not support the background color specified, the background color is not changed.

SMG\$CHANGE_PBD_CHARACTERISTICS

background-color

OpenVMS usage longword_unsigned
type longword (unsigned)
access write only
mechanism by reference

Receives the background color chosen. The **background-color** argument is the address of an unsigned longword into which is written the background color.

This routine may return any of the values listed in the **desired-background-color** argument description or SMG\$C_COLOR_UNKNOWN. If the **desired-background-color** argument is omitted, the value of **background-color** does not change.

Description

SMG\$CHANGE_PBD_CHARACTERISTICS lets you change the width, height, and background color associated with a pasteboard.

If necessary, this routine will notify the OpenVMS operating system of the change in pasteboard characteristics by updating the terminal characteristics displayed when you enter the DCL command SHOW TERMINAL.

Do not use SMG\$CHANGE_PBD_CHARACTERISTICS on a batched pasteboard.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_INVCOLARG	Unknown background color specified.
SMG\$_INVPAGARG	Invalid height of 0 desired.
SMG\$_INVWIDARG	Invalid width of 0 desired.
SMG\$_PBDIN_USE	Cannot change characteristics while batching is on.
SMG\$_WRONUMARG	Wrong number of arguments.
SS\$_xxx	Any error from \$QIOW.

SMG\$CHANGE_RENDITION—Change Default Rendition

The Change Default Rendition routine changes the video attributes for all or part of a virtual display.

Format

```
SMG$CHANGE_RENDITION display-id ,start-row ,start-column ,number-of-rows ,number-of-columns
                    [,rendition-set] [,rendition-complement]
```

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

display-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the virtual display whose default rendition is to be changed. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG\$CREATE_VIRTUAL_DISPLAY.

start-row

OpenVMS usage	longword_signed
type	longword (signed)
access	read only
mechanism	by reference

Starting row position to receive the new rendition. The **start-row** argument is the address of a signed longword that contains the starting row number.

SMG\$CHANGE_RENDITION

start-column

OpenVMS usage longword_signed
type longword (signed)
access read only
mechanism by reference

Starting column position to receive the new rendition. The **start-column** argument is the address of a signed longword that contains the starting column number.

In the case that the right portion of a Chinese character is specified in the **start-column**, the left portion of the character also receives the new rendition.

number-of-rows

OpenVMS usage longword_signed
type longword (signed)
access read only
mechanism by reference

Number of rows to receive the new rendition. The **number-of-rows** argument is the address of a signed longword that contains the number of rows to be affected.

number-of-columns

OpenVMS usage longword_signed
type longword (signed)
access read only
mechanism by reference

Number of columns to receive the new rendition. The **number-of-columns** argument is the address of a signed longword that contains the number of columns to be affected.

rendition-set

OpenVMS usage mask_longword
type longword (unsigned)
access read only
mechanism by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

SMG\$M_BLINK	Displays blinking characters.
SMG\$M_BOLD	Displays characters in higher-than-normal intensity.
SMG\$M_REVERSE	Displays characters in reverse video, that is, using the opposite of the default rendition of the virtual display.
SMG\$M_UNDERLINE	Displays underlined characters.
SMG\$M_INVISIBLE	Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard.
SMG\$M_USER1 through SMG\$M_USER8	Displays user-defined attributes.

The **display-id** argument must be specified when you use the **rendition-set** argument.

rendition-complement

OpenVMS usage mask_longword
 type longword (unsigned)
 access read only
 mechanism by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

Set	Complement	Action
0	0	Attribute set to default
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

Description

This procedure changes the default video rendition of a rectangular block of text already in the specified virtual display. For example, you might use this procedure to redisplay a particular row in reverse video.

In the case Chinese language character set is used, if the **start-column** is on the right portion of a Chinese character, the left portion of the character also receives the new rendition. Also, if the rectangular block ends on the left portion of a Chinese character, the right portion of the character receives the new rendition, too.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_INVROW	Invalid start-row . The specified row is outside the virtual display.
SMG\$_INVCOL	Invalid start-column . The specified column is outside the virtual display.
SMG\$_INVDIS_ID	Invalid display-id .
SMG\$_INVARG	Invalid number of rows, invalid number of columns, unrecognized rendition-set code, or unrecognized rendition-complement code.

SMG\$CHANGE_RENDITION

SMG\$_NO_CHADIS
SMG\$_WRONUMARG

No change in virtual display.
Wrong number of arguments.

SMG\$CHANGE_VIEWPORT—Change the Viewport Associated with a Virtual Display

The Change the Viewport Associated with a Virtual Display routine changes the size of an existing viewport in a virtual display. The text currently in the viewport is remapped to fit the new dimensions.

Format

```
SMG$CHANGE_VIEWPORT display-id [,viewport-row-start] [,viewport-column-start]
                    [,viewport-number-rows] [,viewport-number-columns]
```

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

display-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Identifier of the virtual display containing the viewport to be changed. The **display-id** argument is the address of an unsigned longword containing this identifier.

viewport-row-start

OpenVMS usage	longword_signed
type	longword (signed)
access	read only
mechanism	by reference

Optional row number in the virtual display that will become row 1 in the changed viewport. The **viewport-row-start** argument is the address of a signed longword containing the row number. If omitted, the present **viewport-row-start** value is used.

viewport-column-start

OpenVMS usage	longword_signed
type	longword (signed)
access	read only
mechanism	by reference

Optional column number in the virtual display that will become column 1 in the changed viewport. The **viewport-column-start** argument is the address of a signed longword containing the column number. If omitted, the present **viewport-column-start** value is used.

In the case that the **viewport-column-start** begins on the right portion of a Chinese character, the right portion of the character becomes invisible.

SMG\$CHANGE_VIEWPORT

viewport-number-rows

OpenVMS usage longword_signed
type longword (signed)
access read only
mechanism by reference

Optional number of rows in the changed viewport. The **viewport-number-rows** argument is the address of a signed longword containing the number of rows. If omitted, the present **viewport-number-rows** value is used.

viewport-number-columns

OpenVMS usage longword_signed
type longword (signed)
access read only
mechanism by reference

Optional number of columns in the changed viewport. The **viewport-number-columns** argument is the address of a signed longword containing the number of columns. If omitted, the present **viewport-number-columns** value is used.

Description

SMG\$CHANGE_VIEWPORT lets you change the size of an existing viewport in a virtual display. The text currently in this viewport is remapped to fit the new dimensions, starting at the position specified by the **viewport-row-start** and **viewport-column-start** arguments. This position also specifies the resulting virtual cursor location.

In case Chinese language character set is used, and the **viewport-column-start** begins on the right portion of a Chinese character, the right portion of the character will become invisible. Also if the viewport ends on the left portion of a Chinese character, the left portion of the character will become invisible.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_INVARG	Number of rows or columns is less than zero.
SMG\$_INVCOL	Invalid column specified.
SMG\$_INVDIS_ID	Invalid display-id .
SMG\$_INVROW	Invalid row specified.
SMG\$_NO_WINASSOC	No viewport associated with the virtual display.
SMG\$_WRONUMARG	Wrong number of arguments.

Example

```
C+
C This DEC Fortran example program demonstrates the use of
C SMG$CHANGE_VIEWPORT.
C-

      IMPLICIT INTEGER (A-Z)
      INCLUDE '($SMGDEF)'

C Create the virtual display. Give it a border.

      ROWS = 9
      COLUMNS = 50
```


SMG\$CHANGE_VIEWPORT

```
STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER )
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C Create the pasteboard.

STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C Put data in the virtual display.

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 'This is row 1 in a virtual display with 9 rows.',1,1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 'This is row 2 in a virtual display with 9 rows.',2,1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 'This is row 3 in a virtual display with 9 rows.',3,1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 'This is row 4 in a virtual display with 9 rows.',4,1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 'This is row 5 in a virtual display with 9 rows.',5,1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 'This is row 6 in a virtual display with 9 rows.',6,1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 'This is row 7 in a virtual display with 9 rows.',7,1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 'This is row 8 in a virtual display with 9 rows.',8,1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 'This is row 9 in a virtual display with 9 rows.',9,1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C Paste the virtual display.

STATUS = SMG$COPY_VIRTUAL_DISPLAY(DISPLAY1, DISPLAY2)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$LABEL_BORDER (DISPLAY1, 'Full Display',,, SMG$M_BOLD)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$LABEL_BORDER (DISPLAY2, 'Viewport',,, SMG$M_BOLD)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 2, 10)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY2, PASTE1, 13, 10)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
CALL LIB$WAIT (4.0)

STATUS = SMG$CREATE_VIEWPORT ( DISPLAY2, 2, 1, 5, 21)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
CALL LIB$WAIT (4.0)

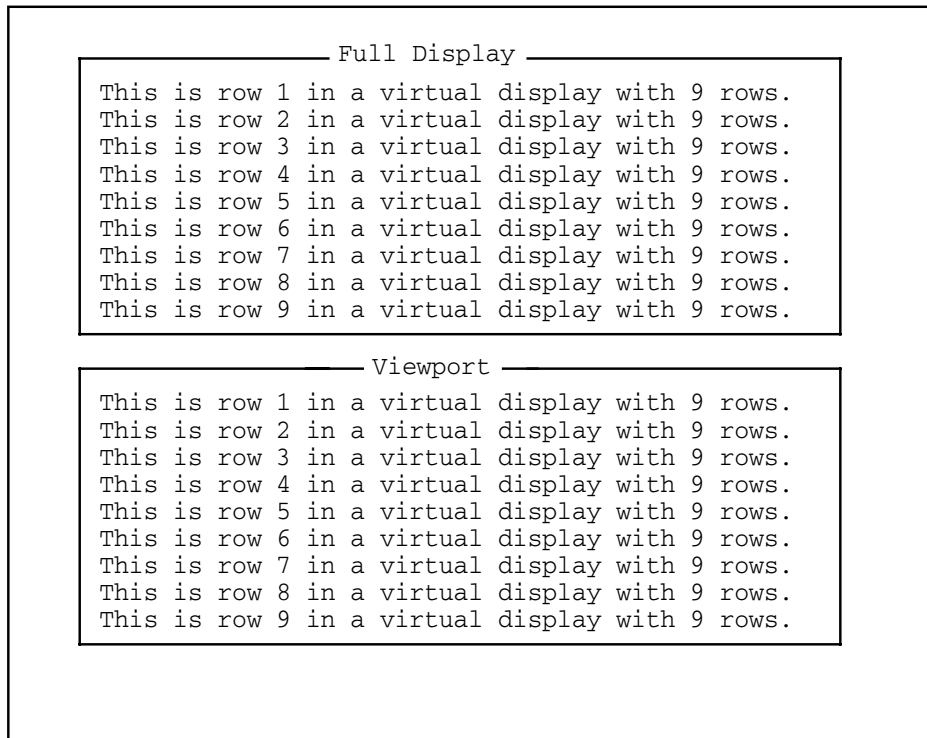
STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY2, PASTE1, 13, 10)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
CALL LIB$WAIT (4.0)
```

SMG\$CHANGE_VIEWPORT

```
STATUS = SMG$CHANGE_VIEWPORT ( DISPLAY2, 4, 8, 3, 15)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
CALL LIB$WAIT (4.0)
END
```

The output generated by this DEC Fortran example is shown in the following figures. In Figure SMG-1, the program has copied the initial virtual display into a second virtual display, labeled “Viewport”.

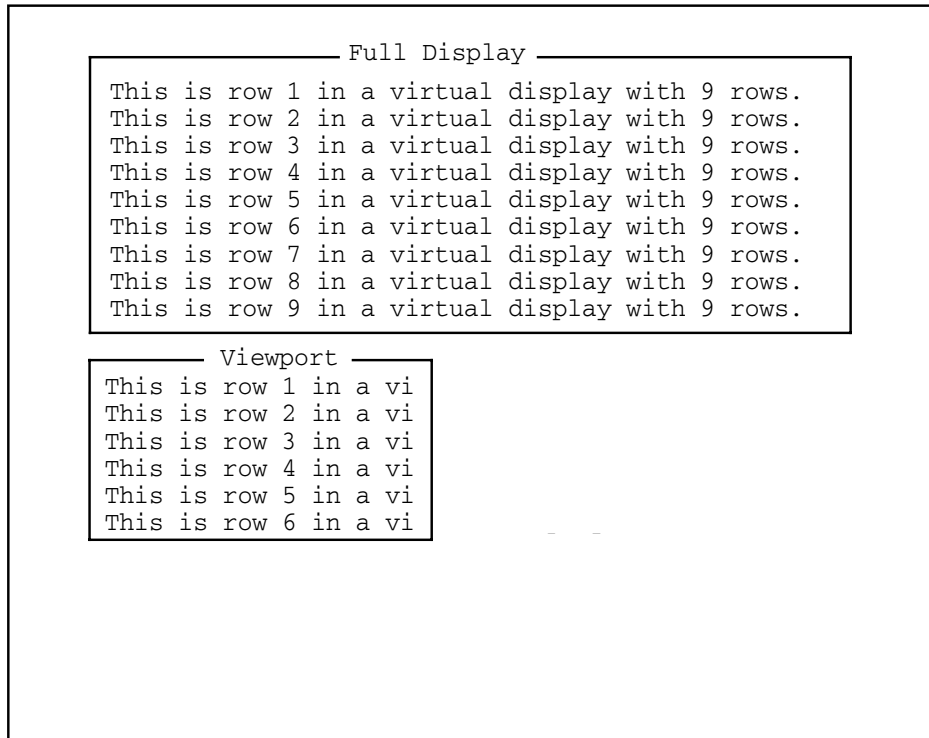
Figure SMG-1 Output Generated After Virtual Displays Are Pasted



ZK-6423/1-GE

After the two identical virtual displays are pasted, the program creates a viewport on the second (copy) virtual display. Once the second display is “repasted”, only the portion located in the viewport is visible. This is shown in Figure SMG-2.

Figure SMG-2 Output Generated After the Viewport Is Created



ZK-6423/2-GE

By calling SMG\$CHANGE_VIEWPORT, the portion of the virtual display that is visible through the viewport is changed. This is shown in Figure SMG-3.

SMG\$CHANGE_VIEWPORT

Figure SMG-3 Output Generated After Calling SMG\$CHANGE_VIEWPORT

```
----- Full Display -----  
This is row 1 in a virtual display with 9 rows.  
This is row 2 in a virtual display with 9 rows.  
This is row 3 in a virtual display with 9 rows.  
This is row 4 in a virtual display with 9 rows.  
This is row 5 in a virtual display with 9 rows.  
This is row 6 in a virtual display with 9 rows.  
This is row 7 in a virtual display with 9 rows.  
This is row 8 in a virtual display with 9 rows.  
This is row 9 in a virtual display with 9 rows.  
  
----- Viewport -----  
row 4 in a vir  
row 5 in a vir  
row 6 in a vir
```

ZK-6423/3-GE

SMG\$CHANGE_VIRTUAL_DISPLAY—Change Virtual Display

The Change Virtual Display routine lets you change the dimensions, border, and video attributes of a virtual display.

Format

```
SMG$CHANGE_VIRTUAL_DISPLAY display-id [,number-of-rows] [,number-of-columns]
                             [,display-attributes] [,video-attributes] [,character-set]
```

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

display-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the virtual display whose attributes are to be changed. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG\$CREATE_VIRTUAL_DISPLAY.

number-of-rows

OpenVMS usage	longword_signed
type	longword (signed)
access	read only
mechanism	by reference

Specifies the new number of rows for the virtual display. The **number-of-rows** argument is the address of a signed longword that contains the number of rows in the virtual display.

number-of-columns

OpenVMS usage	longword_signed
type	longword (signed)
access	read only
mechanism	by reference

Specifies the new number of columns for the virtual display. The **number-of-columns** argument is the address of a signed longword that contains the number of columns in the virtual display.

display-attributes

OpenVMS usage	mask_longword
type	longword (unsigned)
access	read only
mechanism	by reference

SMG\$CHANGE_VIRTUAL_DISPLAY

Specifies the attributes of the virtual display. The **display-attributes** argument is the address of a longword bit mask that contains the display attributes.

Valid values for **display-attributes** are as follows:

SMG\$_BORDER	Specifies a bordered display. If omitted, the display is not bordered.
SMG\$_BLOCK_BORDER	Specifies a block bordered display. If omitted, the display is not bordered.
SMG\$_DISPLAY_CONTROLS	Specifies that control characters such as carriage return and line feed are displayed as graphic characters, if your terminal supports them.
SMG\$_TRUNC_ICON	Specifies that an icon (generally a diamond shape) is displayed where truncation of a line exceeding the width of the virtual display has occurred.

video-attributes

OpenVMS usage	mask_longword
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the default rendition to be applied to all output in a virtual display, unless overridden by a call to a specific output routine. The **video-attributes** argument is the address of an unsigned longword that contains the video attributes mask.

For example, a call to SMG\$PUT_CHARS with an explicit rendition specified would override the default rendition.

The bits that can be set for this argument are as follows:

SMG\$_BLINK	Displays blinking characters.
SMG\$_BOLD	Displays characters in higher-than-normal intensity.
SMG\$_REVERSE	Displays characters in reverse video, that is, to the opposite of the current default rendition of the virtual display.
SMG\$_UNDERLINE	Displays underlined characters.
SMG\$_INVISIBLE	Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard.
SMG\$_USER1 through SMG\$_USER8	Displays user-defined attributes.

Note that you can specify any combination of attributes in a single call. All other bits are reserved for use by Digital and must be zero.

character-set

OpenVMS usage	longword_unsigned
type	longword (unsigned)
access	read only
mechanism	by reference

SMG\$CHANGE_VIRTUAL_DISPLAY

Specifies the default character set for all text in this virtual display. The **character-set** argument is the address of an unsigned longword that contains the character set specifier. The valid values for this argument are as follows.

Value	Character Set
SMG\$C_ASCII	ASCII
SMG\$C_SPEC_GRAPHICS	DEC Special Graphics
SMG\$C_HANYU	DEC Hanyu

Description

SMG\$CHANGE_VIRTUAL_DISPLAY lets you change the size or default attributes of an existing virtual display. If the size of the virtual display is changed, the Screen Management Facility attempts to remap the text associated with the display to fit the new dimensions (starting at row 1 and column 1). If the new size of the virtual display is smaller than the old size, text may be truncated.

In case the truncation starts on the right portion of a Chinese character, the left portion of the character will become an undefined character. If the new size of the virtual display is larger than the old size, text may be padded on the right with spaces.

When a display is redimensioned, the virtual cursor for the display is moved to row 1 and column 1. Note that if a labeled border applies to the virtual display and does not fit the redimensioned display, the label is deleted.

If a program calls both SMG\$CREATE_PASTEBOARD and SMG\$CREATE_VIRTUAL_KEYBOARD, make sure SMG\$CREATE_PASTEBOARD is called first. The program will not function correctly if SMG\$CREATE_VIRTUAL_KEYBOARD is called before SMG\$CREATE_PASTEBOARD.

Original SMG routines support only ASCII(default) and DEC Special Graphics, but Chinese SMG will allow users to specify the **character-set** which enables Chinese language to be used.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
LIB\$_INSVIRMEM	Insufficient virtual memory to reallocate needed buffers.
SMG\$_INVARG	Invalid video or display attributes.
SMG\$_WRONUMARG	Wrong number of arguments.
SMG\$_INVDIS_ID	Invalid display-id .

SMG\$CHECK_FOR_OCCLUSION

SMG\$CHECK_FOR_OCCLUSION—Check for Occlusion

The Check for Occlusion routine checks to see whether a virtual display is covered (occluded) by another virtual display.

Format

SMG\$CHECK_FOR_OCCLUSION display-id ,pasteboard-id ,occlusion-state

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

display-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the virtual display to be checked. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

Display-id is returned by SMG\$CREATE_VIRTUAL_DISPLAY.

pasteboard-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the pasteboard to be checked. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG\$CREATE_PASTEBOARD.

occlusion-state

OpenVMS usage	longword_signed
type	longword (signed)
access	write only
mechanism	by reference

Receives the value denoting whether the display is occluded. The **occlusion-state** argument is the address of a signed longword into which the occlusion state is written. **Occlusion-state** is set to 1 if the display is occluded or set to 0 if the display is not occluded on the specified pasteboard. If the procedure does not return SS\$_NORMAL, the contents of **occlusion-state** are undefined.

Description

SMG\$CHECK_FOR_OCCLUSION determines whether a specified virtual display as pasted to the specified pasteboard is occluded, or covered, by another virtual display.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_NOTPASTED	Specified virtual display is not pasted to the specified pasteboard.
SMG\$_INVPAS_ID	Invalid pasteboard-id .
SMG\$_WRONUMARG	Wrong number of arguments.
SMG\$_INVDIS_ID	Invalid display-id .

Example

```

C+
C This DEC Fortran example program demonstrates the use of
C SMG$CHECK_FOR_OCCLUSION.
C
C This routine creates a virtual display and writes it to the
C pasteboard. Data is placed in the virtual display using SMG$PUT_CHARS.
C-

      INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
      INTEGER SMG$PASTE_VIRTUAL_DISPLAY, SMG$PUT_CHARS
      INTEGER SMG$CHECK_FOR_OCCLUSION
      INTEGER DISPLAY1, DISPLAY2, PASTE1, PASTE2, ROWS, COLUMNS, BORDER
      INTEGER OCCLUSION, STATUS
      CHARACTER*29 TEXT

C+
C Include the SMG definitions. In particular, we want SMG$_BORDER.
C-

      INCLUDE '($SMGDEF)'

C+
C Create two virtual displays using SMG$CREATE_VIRTUAL_DISPLAY.
C Give them borders.
C-

      ROWS = 6
      COLUMNS = 50
      STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1      (ROWS, COLUMNS, DISPLAY1, SMG$_BORDER)
      IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

      ROWS = 5
      COLUMNS = 30
      STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1      (ROWS, COLUMNS, DISPLAY2, SMG$_BORDER)
      IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Create the pasteboard using SMG$CREATE_PASTEBOARD.
C-

      STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
      IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PUT_CHARS to put data into the virtual displays.
C-

      STATUS = SMG$PUT_CHARS ( DISPLAY1,
1      ' This virtual display has 6 rows and 50 columns.', 2, 1)
      IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

```

SMG\$CHECK_FOR_OCCLUSION

```
STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 ' This is a bordered virtual display.', 3, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 4,
1 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 ' This text should be partially occluded.', 5, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY1,
1 ' So should part of this row.', 6, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY2, ' This is virtual', 3, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY2,
1 ' display #2.', 4, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY2,
1 ' This is just some more text.', 5, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-

STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY2, PASTE1, 8, 15)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Check the two virtual displays for occlusion by calling
C SMG$CHECK_FOR_OCCLUSION.
C-

TEXT = 'This display is not occluded.'

STATUS = SMG$CHECK_FOR_OCCLUSION (DISPLAY1, PASTE1, OCCLUSION)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

IF (OCCLUSION .EQ. 0) THEN
    STATUS = SMG$PUT_CHARS (DISPLAY1, TEXT, 1, 1)
    IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
ELSE
    STATUS = SMG$PUT_CHARS (DISPLAY1, 'Occluded.', 1, 1)
    IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
END IF

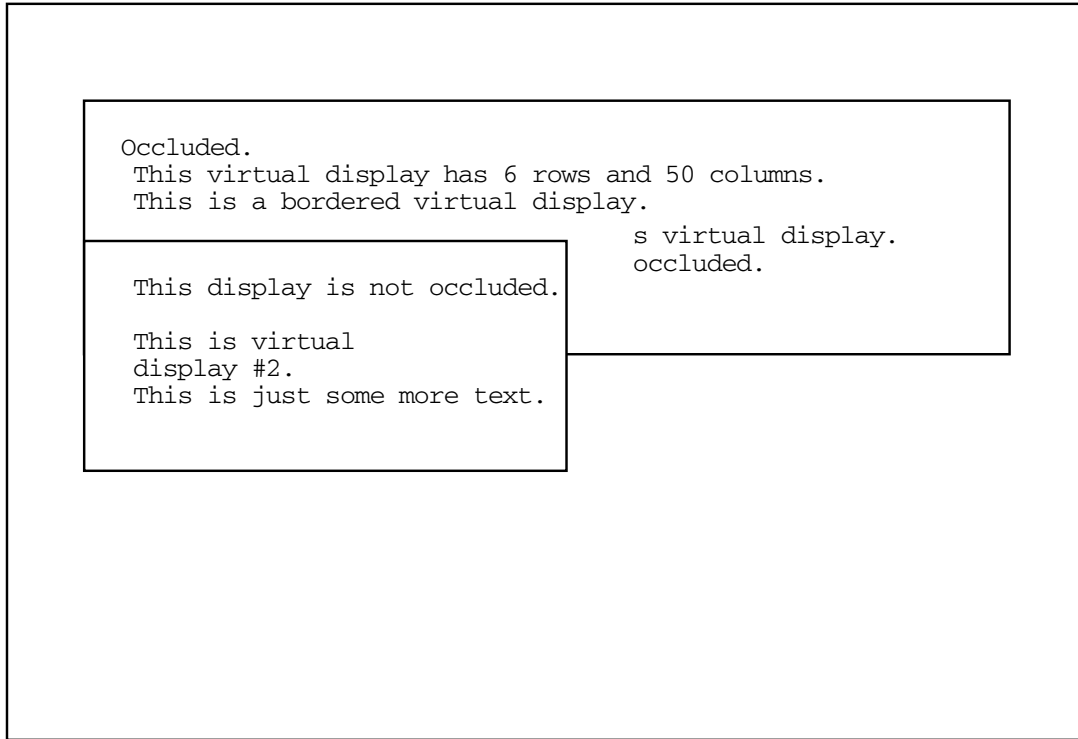
STATUS = SMG$CHECK_FOR_OCCLUSION (DISPLAY2, PASTE1, OCCLUSION)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

IF (OCCLUSION .EQ. 0) THEN
    STATUS = SMG$PUT_CHARS (DISPLAY2, TEXT, 1, 1)
    IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
ELSE
    STATUS = SMG$PUT_CHARS (DISPLAY2, 'Occluded.', 1, 1)
    IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
END IF

END
```

The output generated by this FORTRAN program is shown in Figure SMG-4.

Figure SMG-4 Output Generated by FORTRAN Program Calling SMG\$CHECK_FOR_OCCLUSION



ZK-4128-GE

SMG\$CONTROL_MODE—Control Mode

The Control Mode routine controls the mode of the pasteboard. This includes buffering, minimal updating, whether the screen is cleared when the pasteboard is deleted, and whether tab characters are used for screen formatting.

Format

SMG\$CONTROL_MODE pasteboard-id [,new-mode] [,old-mode] [,buffer-size]

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

pasteboard-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the pasteboard to be changed. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

Pasteboard-id is returned by SMG\$CREATE_PASTEBOARD.

new-mode

OpenVMS usage	mask_longword
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the new control settings to be used. The optional **new-mode** argument is the address of an unsigned longword that contains the mode settings. A bit set to 1 forces that mode to be employed while a bit set to 0 inhibits that mode of operation.

Valid settings are as follows:

SMG\$M_BUF_ENABLED	Enables buffering.
SMG\$M_CLEAR_SCREEN	Causes the Screen Management Facility to clear the screen when the program exits if you have not previously deleted the pasteboard.
SMG\$M_MINUPD	Enables minimal update (the default).
SMG\$M_NOTABS	Causes the Screen Management Facility not to use tab characters to format the screen.
SMG\$M_PROTECT	Protect pasteboard operations from AST interrupts (the default).

All other bits must be zero and are reserved for future use by Digital.

old-mode

OpenVMS usage mask_longword
 type longword (unsigned)
 access write only
 mechanism by reference

Receives the control settings that were in effect before calling this procedure. The optional **old-mode** argument is the address of an unsigned longword into which are written the former mode settings. A bit set to 1 indicates that the specified mode was employed while a bit set to 0 indicates that the mode was inhibited.

buffer-size

OpenVMS usage word_unsigned
 type word (unsigned)
 access read only
 mechanism by reference

Specifies the size of the buffer in bytes. The optional **buffer-size** argument is the address of an unsigned word that contains the size of the buffer. **Buffer-size** is used when buffering mode is enabled (SMG\$M_BUF_ENABLED). The default and minimum buffer size is 256 bytes. The maximum value is 65535. The **buffer-size** value depends on UAF values and is maximized with the sysgen parameter MAXBUF.

Description

SMG\$CONTROL_MODE lets you determine and change the mode of the Screen Management Facility operation for a specified pasteboard. By specifying different combinations of the **new-mode** and **old-mode** arguments, SMG\$CONTROL_MODE can be used in various ways.

- To use SMG\$CONTROL_MODE to determine the current mode settings, use the following format:

```
SMG$CONTROL_MODE (pasteboard_id ,old_mode)
```

- To use SMG\$CONTROL_MODE to set the bits without regard to their current setting, use the following format:

```
SMG$CONTROL_MODE (pasteboard_id ,new_mode)
```

- To use SMG\$CONTROL_MODE to save the current settings, set new modes, and later restore the original settings, use the following format:

```
SMG$CONTROL_MODE (pasteboard_id ,new_mode ,old_mode)
```

This retrieves the current bit settings and then sets the mode according to the **new-mode** argument.

Later, to restore the mode to its former state, specify the following format:

```
SMG$CONTROL_MODE (pasteboard_id ,old_mode)
```

This sets the new mode according to the settings previously retrieved.

Note that if both arguments are omitted, no information is returned.

The modes that can be determined and changed using SMG\$CONTROL_MODE are as follows:

Buffering

SMG\$CONTROL_MODE

In this mode, the Screen Management Facility buffers all output for efficient use of system QIOs. When the buffer fills, SMG\$ writes the buffer to the terminal. By calling SMG\$FLUSH_BUFFER, the user can force to the screen any output that has been placed in the pasteboard buffer but not yet written to the terminal.

Minimal Screen Update

By default, the Screen Management Facility tries to minimize the number of characters actually sent to the terminal. It does this by keeping track of the current contents of the pasteboard and the new contents of the pasteboard. SMG\$ then sends only those characters that have changed.

Nonminimal updating rewrites any line containing a change, starting with the first changed character on that line.

Clear Screen

By default, the Screen Management Facility does not clear the screen when the program exits if you have not already deleted the pasteboard. Use the clear screen mode to prevent this default behavior.

No Tabs

If this bit is set, the Screen Management Facility does not rely on the terminal's tab settings. If it is not set, the Screen Management Facility will use physical tabs for the minimal update procedure. However, note that such use implicitly assumes that the tab stops are set to the Digital default locations (every eight characters). Specify "no tabs" if you want to be sure that the application will run regardless of the tab settings the user has set on the terminal. By default, this bit is clear. A terminal setting of SET TERM/NOTABS may also be used to override this default.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_INVARG	Invalid argument. New-mode has a bit set which does not correspond to SMG\$_BUF_ENABLED, SMG\$_MINUPD, SMG\$_CLEAR_SCREEN, or SMG\$_NOTABS, or buffer size is less than 256.
SMG\$_INVPAS_ID	Invalid pasteboard-id .
SMG\$_WRONUMARG	Wrong number of arguments.

SMG\$COPY_VIRTUAL_DISPLAY—Copy a Virtual Display

The Copy a Virtual Display routine creates a copy of an existing virtual display and assigns to it a new virtual display identifier.

Format

```
SMG$COPY_VIRTUAL_DISPLAY current-display-id ,new-display-id
```

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

current-display-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Display identifier of the virtual display to be replicated. The **current-display-id** argument is the address of the unsigned longword that contains the display identifier.

new-display-id

OpenVMS usage	identifier
type	longword (unsigned)
access	write only
mechanism	by reference

Receives the display identifier of the newly created virtual display. The **new-display-id** argument is the address of the unsigned longword that receives the new display identifier.

Description

SMG\$COPY_VIRTUAL_DISPLAY creates a copy of an existing virtual display and assigns to it a new virtual display number. This newly created virtual display will not be pasted anywhere; use SMG\$PASTE_VIRTUAL_DISPLAY and the **new-display-id** identifier to paste the newly created virtual display. The existing display being replicated does not have to be pasted when SMG\$COPY_VIRTUAL_DISPLAY is invoked.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
LIB\$_INSVIRMEM	Insufficient virtual memory to allocate needed buffer.

SMG\$COPY_VIRTUAL_DISPLAY

Example

```
C+
C This DEC Fortran example program demonstrates the use of
C SMG$COPY_VIRTUAL_DISPLAY.
C
C This routine creates a virtual display and writes it to the
C pasteboard. Data is placed in the virtual display using SMG$PUT_CHARS.
C-

        IMPLICIT INTEGER (A-Z)
        CHARACTER*29 TEXT

C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-

        INCLUDE '($SMGDEF)'

C+
C Create two virtual displays using SMG$CREATE_VIRTUAL_DISPLAY.
C Give them borders.
C-

        ROWS = 6
        COLUMNS = 50

        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1         (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

        ROWS = 5
        COLUMNS = 30

        STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1         (ROWS, COLUMNS, DISPLAY2, SMG$M_BORDER)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Create the pasteboard using SMG$CREATE_PASTEBOARD.
C-

        STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PUT_CHARS to put data into the virtual displays.
C-

        STATUS = SMG$PUT_CHARS ( DISPLAY1,
1         ' This virtual display has 6 rows and 50 columns.', 2, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

        STATUS = SMG$PUT_CHARS ( DISPLAY1,
1         ' This is a bordered virtual display.', 3, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

        STATUS = SMG$PUT_CHARS ( DISPLAY1,
1         ' SMG$PUT_CHARS puts data in this virtual display.', 4,
1         1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

        STATUS = SMG$PUT_CHARS ( DISPLAY1,
1         ' This text should be partially occluded.', 5, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

        STATUS = SMG$PUT_CHARS ( DISPLAY1,
1         ' So should part of this row.', 6, 1)
        IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```


SMG\$COPY_VIRTUAL_DISPLAY

```
STATUS = SMG$PUT_CHARS ( DISPLAY2, ' This is virtual', 3, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY2,
1 ' display #2.', 4, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS ( DISPLAY2,
1 ' This is just some more text.', 5, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-

STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY1, PASTE1, 4, 15)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY2, PASTE1, 8, 15)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Copy the first virtual display, the one that is partially occluded.
C-

STATUS = SMG$COPY_VIRTUAL_DISPLAY ( DISPLAY1, NEW_DISPLAY)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Now paste this new virtual display so that it occludes the other displays.
C-

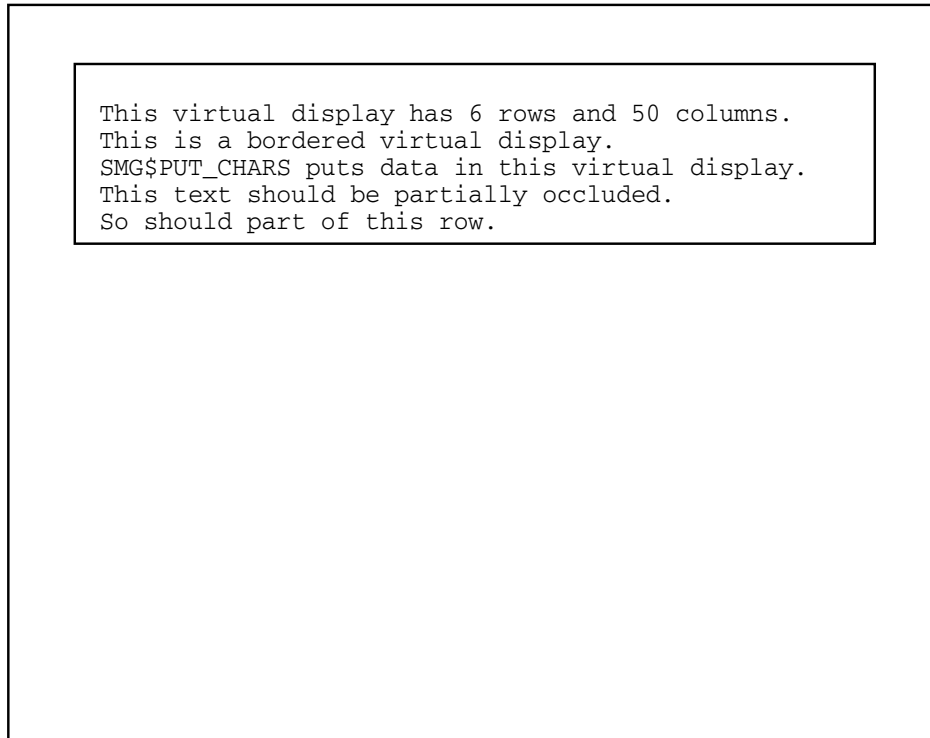
STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( NEW_DISPLAY, PASTE1, 4, 20)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

END
```

The first virtual display created by this FORTRAN example is shown in Figure SMG-5.

SMG\$COPY_VIRTUAL_DISPLAY

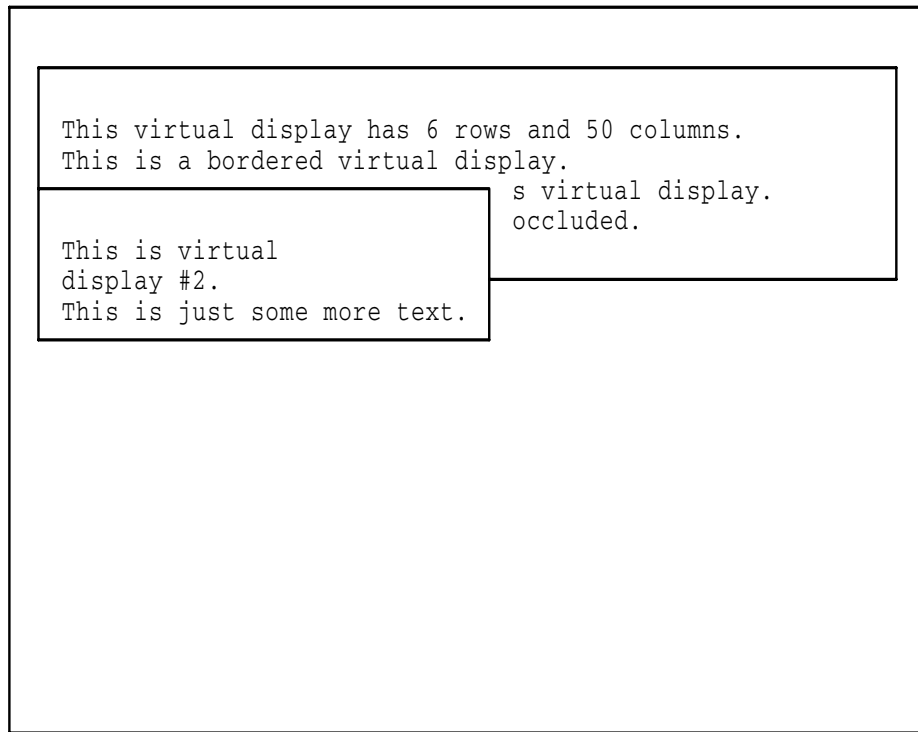
Figure SMG-5 First Virtual Display Generated by This Example



ZK-4808-GE

The second virtual display created by this FORTRAN example is shown in Figure SMG-6.

Figure SMG-6 Second Virtual Display Generated by This Example

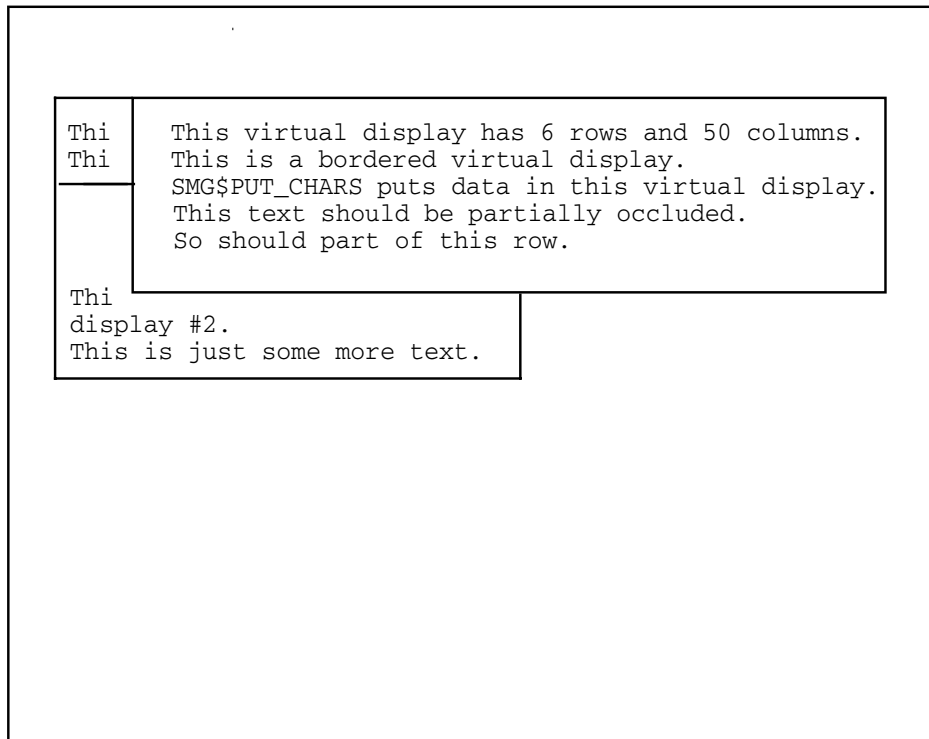


ZK-4809-GE

The output generated after the call to SMG\$COPY_VIRTUAL_DISPLAY is shown in Figure SMG-7.

SMG\$COPY_VIRTUAL_DISPLAY

Figure SMG-7 Output Generated After the Call to SMG\$COPY_VIRTUAL_DISPLAY



ZK-4810-GE

SMG\$CREATE_KEY_TABLE—Create Key Table

The Create Key Table routine creates a table for key definitions.

Format

SMG\$CREATE_KEY_TABLE key-table-id

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

key-table-id	
OpenVMS usage	identifier
type	longword (unsigned)
access	write only
mechanism	by reference

Receives the identifier of the newly created key table. The **key-table-id** argument is the address of an unsigned longword into which is written the key table identifier.

Description

SMG\$CREATE_KEY_TABLE creates a key definition table. Key definitions can then be added to this table with the SMG\$ADD_KEY_DEF, SMG\$LOAD_KEY_DEFS, and SMG\$DEFINE_KEY routines, and used with the SMG\$READ_COMPOSED_LINE routine.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_WRONUMARG	Wrong number of arguments.
LIB\$_INSVIRMEM	Insufficient virtual memory.

SMG\$CREATE_MENU—Fill the Virtual Display with a Menu

The Fill the Virtual Display with a Menu routine displays menu choices in the virtual display indicated, starting at the specified row.

Format

```
SMG$CREATE_MENU display-id ,choices [,menu-type] [,flags] [,row] [,rendition-set]
                [,rendition-complement]
```

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

display-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Display identifier of the virtual display in which the menu is created. The **display-id** argument is the address of an unsigned longword containing this identifier.

choices

OpenVMS usage	static array of char_string
type	character string
access	read only
mechanism	by descriptor

Static array in which each element corresponds to an item to be displayed in the menu. The **choices** argument is the address of a descriptor pointing to this static array of character strings. Note that blank menu items are ignored.

menu-type

OpenVMS usage	longword_unsigned
type	longword (unsigned)
access	read only
mechanism	by reference

Optional bit mask specifying the type of menu to be displayed. The **menu-type** argument is the address of a longword bit mask that specifies this menu type. Valid values are as follows:

SMG\$K_BLOCK	The menu items are displayed in matrix format (default).
SMG\$K_VERTICAL	Each menu item is displayed on its own line.
SMG\$K_HORIZONTAL	The menu items are displayed all on one line.

flags

OpenVMS usage mask_longword
 type longword (unsigned)
 access read only
 mechanism by reference

Optional bit mask specifying the attributes to be used for the menu. The **flags** argument is the address of an unsigned longword that contains the flag. Valid values are as follows:

SMG\$_DOUBLE_SPACE	Double-spaced rows of menu items. The default is single spaced.
SMG\$_FIXED_FORMAT	Each menu item is in a fixed-length field. The field is the size of the largest menu item. The default is compress.
SMG\$_WIDE_MENU	Wide characters are used in the menu items. The default is normal sized characters.

row

OpenVMS usage longword_signed
 type longword (signed)
 access read only
 mechanism by reference

Optional row number in the specified virtual display at which the first menu item is displayed. The **row** argument is the address of a signed longword that contains this row number. If **row** is omitted, the first row of the virtual display's scrolling region is used.

rendition-set

OpenVMS usage mask_longword
 type longword (unsigned)
 access read only
 mechanism by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be used when writing out the menu choices. The following attributes can be specified using the **rendition-set** argument:

SMG\$_BLINK	Displays blinking characters.
SMG\$_BOLD	Displays characters in higher-than-normal intensity.
SMG\$_REVERSE	Displays characters in reverse video, that is, using the opposite of the default rendition of the virtual display.
SMG\$_UNDERLINE	Displays underlined characters.
SMG\$_USER1 through SMG\$_USER8	Displays user-defined attributes.

The **display-id** argument must be specified when you use the **rendition-set** argument.

rendition-complement

OpenVMS usage mask_longword
 type longword (unsigned)
 access read only

SMG\$CREATE_MENU

mechanism by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

Set	Complement	Action
0	0	Attribute set to default
1	0	Attribute on
0	1	Attribute set to complement of default setting
1	1	Attribute off

Description

SMG\$CREATE_MENU displays a list of menu choices in the virtual display's virtual scrolling region, starting in a specified row. Menu items start in the second column of the virtual display. Multiple menu items on the same row are separated by four spaces.

The choices are displayed with the specified rendition attributes in any one of the following formats:

Vertical Each menu item is on its own line.
Horizontal The menu items are all on one line.
Block The menu items appear in matrix format.

Any menu items that do not fit within the bounds of the virtual display are not displayed, but are saved for later scrolling by SMG\$SELECT_FROM_MENU. The choices will be single spaced by default, but if requested this can be changed to double spaced. Four spaces separate each menu item horizontally. If requested, the items can also be displayed in fixed format columns where the width of the column is equal to the size of the largest string passed.

After a call to SMG\$CREATE_MENU, the user must not output any characters to the display that will disturb the rows containing the newly created menu. If characters are output that do interfere with the menu, unpredictable results will be generated. Use the SMG\$SELECT_FROM_MENU routine to select an item from this menu.

SMG\$CREATE_MENU supports one-dimensional NCA and VSA string arrays. NCA and VSA string arrays include all string arrays generated by VAX Ada, VAX BASIC, VAX DIBOL, DEC Fortran, VAX Pascal, and VAX PL/I. (Note that if you are calling SMG\$CREATE_MENU from VAX BASIC you must still use a MAP statement to declare the array.)

The character set attribute of the menu items are decided by that of the virtual display on which the menu items are displayed.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_INVDIS_ID	The display identifier is invalid or contains a viewport.
LIB\$_xxxx	Any condition value returned by LIB\$CREATE_VM_ZONE, LIB\$GET_VM, and LIB\$FREE_VM.
SMG\$_xxxx	Any condition value returned by SMG\$PUT_CHARS, SMG\$BEGIN_DISPLAY_UPDATE, and SMG\$END_DISPLAY_UPDATE.

Example

```
#module menu "001"
/*****
/* INCLUDE FILES */
*****/

#include <descrip.h>
#include <smgdef.h>

/*****
/* EXTERNAL DEFINITIONS */
*****/

globalvalue SS$_NORMAL;

extern int SMG$CREATE_MENU();
extern int SMG$SELECT_FROM_MENU();
extern int SMG$CREATE_PASTEBOARD();
extern int SMG$CREATE_VIRTUAL_DISPLAY();
extern int SMG$CREATE_VIRTUAL_KEYBOARD();
extern int SMG$PASTE_VIRTUAL_DISPLAY();
extern int SMG$PUT_LINE();
extern int LIB$SIGNAL();
extern int LIB$SFREE1_DD();

/*****
/* MACRO DEFINITIONS */
*****/

#define K_NUM_STRINGS 13

/*****
/* MAIN PROGRAM */
*****/

int
main (argc, argv)

unsigned int  argc;
char        **argv;
{

struct dsc$descriptor_s s_desc;
struct dsc$descriptor_a array_desc;
int      pbid, kbid, did, did1, sts, s, a_size, n = 0, max_len = 0;
char     *array, *p;
char     *months[K_NUM_STRINGS] = {"January", "February", "March",
                                     "April", "May", "June", "July", "August", "September",
                                     "October", "November", "December", "Exit menu"};
```

SMG\$CREATE_MENU

```
/* Set up the string descriptor for the returned menu item */
s_desc.dsc$w_length = 0;
s_desc.dsc$b_dtype = DSC$K_DTYPE_T;
s_desc.dsc$b_class = DSC$K_CLASS_D;
s_desc.dsc$a_pointer = 0;

/* find length of largest string */
for (s=0; s<K_NUM_STRINGS; s++)
    max_len = max_len > strlen(months[s]) ? max_len : strlen(months[s]);

/* put the menu choices in vm */
a_size = max_len * K_NUM_STRINGS;
p = array = malloc(a_size);
for (s=0; s<K_NUM_STRINGS; s++)
    {
        strncpy(p, months[s], max_len);
        p += max_len;
    };

/* Set up the string array descriptor for the menu choices */
array_desc.dsc$w_length = max_len;
array_desc.dsc$b_dtype = DSC$K_DTYPE_T;
array_desc.dsc$b_class = DSC$K_CLASS_A;
array_desc.dsc$a_pointer = array;
array_desc.dsc$b_aflags.dsc$v_fl_bounds = 0;
array_desc.dsc$b_dimct = 1;
array_desc.dsc$l_arsize = a_size;

/* set up the pasteboard, keyboard, and virtual displays */
sts = SMG$CREATE_PASTEBOARD(&pbid);
if (sts != SS$NORMAL) s = LIB$SIGNAL(sts);

sts = SMG$CREATE_VIRTUAL_KEYBOARD(&kbid);
if (sts != SS$NORMAL) s = LIB$SIGNAL(sts);

sts = SMG$CREATE_VIRTUAL_DISPLAY(&6, &50, &did, &SMG$M_BORDER);
if (sts != SS$NORMAL) s = LIB$SIGNAL(sts);

sts = SMG$CREATE_VIRTUAL_DISPLAY(&6, &50, &did1, &SMG$M_BORDER);
if (sts != SS$NORMAL) s = LIB$SIGNAL(sts);

sts = SMG$PASTE_VIRTUAL_DISPLAY(&did, &pbid, &2, &2);
if (sts != SS$NORMAL) s = LIB$SIGNAL(sts);

sts = SMG$PASTE_VIRTUAL_DISPLAY(&did1, &pbid, &10, &2);
if (sts != SS$NORMAL) s = LIB$SIGNAL(sts);

/* set up the menu */
sts = SMG$CREATE_MENU(&did1, &array_desc, &SMG$K_BLOCK,
                    &SMG$M_DOUBLE_SPACE, 0, &SMG$M_BOLD);
if (sts != SS$NORMAL) s = LIB$SIGNAL(sts);
```

SMG\$CREATE_MENU

```
while (n<13)
{
  sts = SMG$SELECT_FROM_MENU(&kbid, &did1, &n, 0, 0, 0, 0, &s_desc);
  if (sts != SS$NORMAL) s = LIB$SIGNAL(sts);

  sts = SMG$PUT_LINE(&did, &s_desc);
  if (sts != SS$NORMAL) s = LIB$SIGNAL(sts);
}

/*****
/* Free unneeded storage */
*****/

free(array);
sts = LIB$SFREE1_DD(&s_desc);
if (sts != SS$NORMAL) s = LIB$SIGNAL(sts);

return;
}
```

SMG\$CREATE_PASTEBOARD

SMG\$CREATE_PASTEBOARD—Create a Pasteboard

The Create Pasteboard routine creates a pasteboard and returns its assigned pasteboard identifier.

Format

```
SMG$CREATE_PASTEBOARD  pasteboard-id [,output-device] [,number-of-pasteboard-rows]
                        [,number-of-pasteboard-columns] [,flags] [,type-of-terminal]
                        [,device-name]
```

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

pasteboard-id

OpenVMS usage	identifier
type	longword (unsigned)
access	write only
mechanism	by reference

Receives the identifier of the newly created pasteboard. The **pasteboard-id** argument is the address of an unsigned longword into which is written the new pasteboard identifier.

output-device

OpenVMS usage	device_name
type	character string
access	read only
mechanism	by descriptor

Specifies the file specification or logical name to which the output associated with this pasteboard will be written. The **output-device** argument is the address of a descriptor that points to the name of the output device. If omitted, output is sent to SYS\$OUTPUT.

Note

An RMS file is not supported as an output device in Chinese Screen Management Facility.

number-of-pasteboard-rows

OpenVMS usage	longword_signed
type	longword (signed)
access	write only
mechanism	by reference

Receives the number of rows on the device specified in the **output-device** argument. The **number-of-pasteboard-rows** argument is the address of a

signed longword into which is written the number of rows on the specified device, which will be the number of rows in the pasteboard.

number-of-pasteboard-columns

OpenVMS usage	longword_signed
type	longword (signed)
access	write only
mechanism	by reference

Receives the number of columns on the device specified in the **output-device** argument. The **number-of-pasteboard-columns** argument is the address of a signed longword into which is written the number of columns on the specified device.

flags

OpenVMS usage	mask_longword
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the attributes to be used in the pasteboard. The **flags** argument is the address of an unsigned longword that contains the flag. The default action is to clear the screen when the pasteboard is created.

Valid value is:

SMG\$M_KEEP_CONTENTS	The screen is not initially cleared. The Screen Management Facility works best when it can manage the entire screen. Therefore, using SMG\$M_KEEP_CONTENTS is discouraged.
----------------------	--

type-of-terminal

OpenVMS usage	mask_longword
type	longword (unsigned)
access	write only
mechanism	by reference

Receives the SMG\$ internal device type to which the output associated with this pasteboard will be written. The **type-of-terminal** argument is the address of an unsigned longword into which is written the terminal type.

The returned values are as follows:

```
SMG$K_UNKNOWN
SMG$K_VTFORIGN
SMG$K_HARDCOPY
SMG$K_VTTERMTABLE
```

If a value other than SMG\$K_VTTERMTABLE is returned, you must use SMG\$SNAPSHOT to output the contents of the pasteboard.

Note

SMG\$SNAPSHOT is not supported in Chinese Screen Management Facility.

SMG\$CREATE_PASTEBOARD

Description

SMG\$CREATE_PASTEBOARD creates a new pasteboard, associates it with the device specified by **output-device**, and returns its assigned **pasteboard-id**. Note that if you request a pasteboard on a device that already has a pasteboard assigned, this routine returns the **pasteboard-id** of the existing pasteboard and returns the SMG\$_PASALREXI status code.

If a program calls both SMG\$CREATE_PASTEBOARD and SMG\$CREATE_VIRTUAL_KEYBOARD, make sure SMG\$CREATE_PASTEBOARD is called first. The program will not function correctly if SMG\$CREATE_VIRTUAL_KEYBOARD is called before SMG\$CREATE_PASTEBOARD.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_PASALREXI	Successful completion. A pasteboard already exists for this device.
SMG\$_WRONUMARG	Wrong number of arguments.
LIB\$_INSVIRMEM	Insufficient virtual memory to allocate needed buffer.

Any condition values returned by LIB\$GET_EF, LIB\$GET_VM, \$QIO, \$GETDVI, \$ASSIGN.

```
0 | 1 | 2 | 3 | 4 | 5 | 6 |
123456789012345678901234567890123456789012345678901234567890
```

Examples

```
1.
C* RPG example for SMG$CREATE_PASTEBOARD.
C      CREPAS  EXTRN 'SMG$CREATE_PASTEBOARD'
C      CREDIS  EXTRN 'SMG$CREATE_VIRTUAL_DISPLAY'
C      PUTCHA  EXTRN 'SMG$PUT_CHARS'
C      PASDIS  EXTRN 'SMG$PASTE_VIRTUAL_DISPLAY'
C      Z-ADD0      ZERO      90
C      Z-ADD1      LINCOL    90
C      Z-ADD2      LINE      90
C      Z-ADD5      COLUMN    90
C      MOVE 'Menu'  OUT       4
C* Create the pasteboard.
C      CALL CREPAS
C      PARM          PASTID  90 WL
C      PARMV        ZERO
C      PARM          HEIGHT  90 WL
C      PARM          WIDTH  90 WL
C* Create the virtual display.
C      CALL CREDIS
C      PARM          HEIGHT   RL
C      PARM          WIDTH   RL
C      PARM          DISPID  90 WL
C* Output the 'Menu'.
C      CALL PUTCHA
C      PARM          DISPID   RL
C      PARMD        OUT
C      PARM          LINE     RL
C      PARM          COLUMN   RL
C* Paste the virtual display.
C      CALL PASDIS
```

SMG\$CREATE_PASTEBOARD

C	PARM	DISPID	RL
C	PARM	PASTID	RL
C	PARM	LINCOL	RL
C	PARM	LINCOL	RL
C	SETON		LR

The RPG II program above displays 'Menu' beginning at line 2, column 5.

```
2. C+ SMG1.FOR
C This DEC Fortran example program demonstrates the use of
C SMG$CREATE_PASTEBOARD.
C-

      IMPLICIT INTEGER*4 (A-Z)
      SMG$M_BOLD = 1
      SMG$M_REVERSE = 2
      SMG$M_BLINK = 4
      SMG$M_UNDERLINE = 8

C+
C Establish the terminal screen as a pasteboard
C by calling SMG$CREATE_PASTEBOARD.
C-

      STATUS = SMG$CREATE_PASTEBOARD (NEW_PID,,,)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))

C+
C Establish a virtual display region by calling
C SMG$CREATE_VIRTUAL_DISPLAY.
C-

      STATUS = SMG$CREATE_VIRTUAL_DISPLAY (5,80,DISPLAY_ID,,)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))

C+
C Paste the virtual display to the screen, starting at
C row 10, column 15 using SMG$PASTE_VIRTUAL_DISPLAY.
C-

      STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY_ID,NEW_PID,10,15)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))

C+
C Write three lines to the screen using SMG$PUT_LINE.
C-

      STATUS = SMG$PUT_LINE (DISPLAY_ID,'This line is underlined',2,
1                               SMG$M_UNDERLINE,0,,)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))

      STATUS = SMG$PUT_LINE (DISPLAY_ID,'This line is blinking',2,
1                               SMG$M_BLINK,0,,)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))

      STATUS = SMG$PUT_LINE (DISPLAY_ID,'This line is reverse video',2,
1                               SMG$M_REVERSE,0,,)
      IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))

      END
```

This FORTRAN program calls Run-Time Library Screen Management routines to format screen output.

SMG\$CREATE_SUBPROCESS

SMG\$CREATE_SUBPROCESS—Create and Initialize a Subprocess

The Create and Initialize a Subprocess routine creates a DCL subprocess and associates it with a virtual display.

Format

SMG\$CREATE_SUBPROCESS display-id [,AST-routine] [,AST-argument]

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

display-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

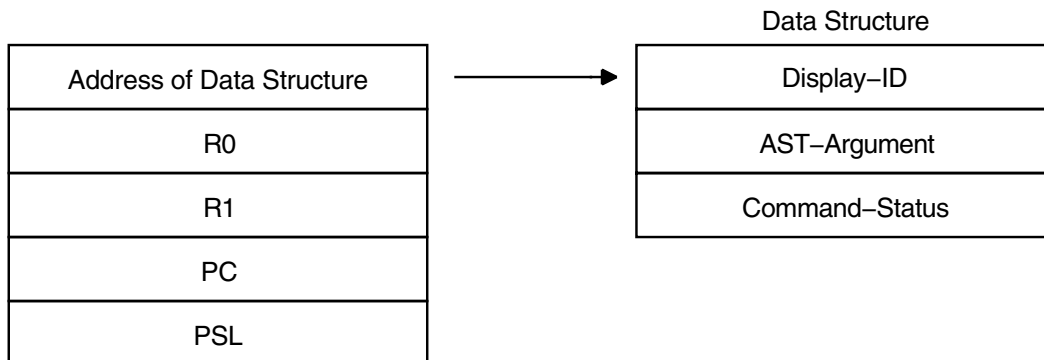
Identifier of the virtual display with which the newly created subprocess is associated. The **display-id** argument is the address of an unsigned longword containing this identifier.

AST-routine

OpenVMS usage	ast_procedure
type	procedure value
access	call without stack unwinding
mechanism	by value

Optional AST routine to be called when the currently executing command completes. The **AST-routine** argument is the routine's procedure entry mask.

The AST routine is called with five parameters. The first parameter is a pointer to a data structure that contains the **display-id**, **AST-argument**, and the **command-status** values. The remaining four parameters for the AST routine are R0, R1, PC, and PSL.



ZK-6508-GE

If the **AST-routine** argument is specified, the routine SMG\$EXECUTE_COMMAND buffers any commands passed to it and executes them in order, calling the specified AST routine when each command completes. If the **AST-routine** argument is not specified, SMG\$EXECUTE_COMMAND waits until the specified command completes before returning control to the user.

AST-argument

OpenVMS usage	user_arg
type	longword (unsigned)
access	read only
mechanism	by value

Optional argument you supply to the AST routine. The **AST-argument** parameter is an unsigned longword that contains the value to be passed to the AST routine.

Description

SMG\$CREATE_SUBPROCESS lets you create a DCL subprocess and associate this subprocess with a virtual display. (The subprocess is initialized using the SET NOVERIFY and SET NOON DCL commands.) From your main process you can then specify commands to be executed by the subprocess using the SMG\$EXECUTE_COMMAND routine. Communication between processes is performed using mailboxes, thus allowing you to control the input commands and the output text. When buffering commands, use the optional AST routine to notify your main process whenever a command is completed. Broadcast trapping and unsolicited input do not have to be disabled to use this routine.

Before creating the subprocess, the Screen Management Facility checks to ensure that you have sufficient resources to create the necessary mailboxes and the subprocess. A remaining BYTLM value of at least 5000 and a remaining PRCLM value of at least 1 are required.

The Screen Management Facility declares an exit handler that deletes the subprocess if the user exits without first calling the routine SMG\$DELETE_SUBPROCESS. Under some circumstances, however, these facility-supplied exit handlers are not executed. In this case, you must delete the subprocess with the DCL SHOW PROCESS/SUB command followed by the DCL STOP command.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_SUBALREXI	Subprocess already exists for this display-id (alternate success status).
SMG\$_INSQUOCRE	Insufficient quota remaining to create subprocess.
SMG\$_INVDIS_ID	Invalid display-id .
SS\$_xxxx	Any status from \$GETDVI, \$GETJPI, \$DCLEXH, or \$CREMBX.
LIB\$_xxxx	Any status from LIB\$SPAWN, LIB\$GET_EF, or LIB\$GET_VM.

SMG\$CREATE_SUBPROCESS

Example

```
10
!
! This VAX BASIC program demonstrates the use of
! SMG$CREATE_SUBPROCESS.
!

OPTION TYPE = EXPLICIT
OPTION CONSTANT TYPE = INTEGER

%INCLUDE "LIB$ROUTINES" %FROM %LIBRARY "SYS$LIBRARY: BASIC$STARLET"
%INCLUDE "SMG$ROUTINES" %FROM %LIBRARY "SYS$LIBRARY: BASIC$STARLET"      !***
%INCLUDE "$SMGDEF" %FROM %LIBRARY "SYS$LIBRARY: BASIC$STARLET"
%INCLUDE "$SSDEF" %FROM %LIBRARY "SYS$LIBRARY: BASIC$STARLET"

COMMON LONG NUM_COMMANDS

DECLARE SMG$R_SUBPROCESS_INFO_TABLE SMG_INFO                               !***
DECLARE LONG S, PASTEBOARD_ID, DISPLAY_ID, STATUS_DISPLAY_ID
EXTERNAL INTEGER COMPLETION_ROUTINE                                       !***

S = SMG$CREATE_PASTEBOARD (PASTEBOARD_ID)
IF S <> SS$NORMAL THEN CALL LIB$SIGNAL (S) END IF

S = SMG$CREATE_VIRTUAL_DISPLAY (12,75,DISPLAY_ID,SMG$M_BORDER)
IF S <> SS$NORMAL THEN CALL LIB$SIGNAL (S) END IF

S = SMG$CREATE_VIRTUAL_DISPLAY (5,75,STATUS_DISPLAY_ID,SMG$M_BORDER)
IF S <> SS$NORMAL THEN CALL LIB$SIGNAL (S) END IF

S = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY_ID,PASTEBOARD_ID,2,2)
IF S <> SS$NORMAL THEN CALL LIB$SIGNAL (S) END IF

S = SMG$PASTE_VIRTUAL_DISPLAY (STATUS_DISPLAY_ID,PASTEBOARD_ID,17,2)
IF S <> SS$NORMAL THEN CALL LIB$SIGNAL (S) END IF

S = SMG$CREATE_SUBPROCESS (DISPLAY_ID,      &
    LOC(COMPLETION_ROUTINE), &
    STATUS_DISPLAY_ID)
IF S <> SS$NORMAL THEN CALL LIB$SIGNAL (S) END IF

NUM_COMMANDS = 1
S = SMG$EXECUTE_COMMAND(DISPLAY_ID, "$SHOW DEFAULT")
IF S <> SS$NORMAL THEN CALL LIB$SIGNAL (S) END IF

NUM_COMMANDS = NUM_COMMANDS + 1
S = SMG$EXECUTE_COMMAND(DISPLAY_ID, "$SHOW TIME")
IF S <> SS$NORMAL THEN CALL LIB$SIGNAL (S) END IF

NUM_COMMANDS = NUM_COMMANDS + 1
S = SMG$EXECUTE_COMMAND(DISPLAY_ID, "$SHOW QUOTA")
IF S <> SS$NORMAL THEN CALL LIB$SIGNAL (S) END IF
SLEEP (5) UNTIL NUM_COMMANDS <= 0
END

20 SUB COMPLETION_ROUTINE(SMG$R_SUBPROCESS_INFO_TABLE SMG_INFO, & !***
    LONG R0, LONG R1, LONG PC, LONG PSL)
OPTION TYPE = EXPLICIT
OPTION CONSTANT TYPE = INTEGER
%INCLUDE "$SMGDEF" %FROM %LIBRARY "SYS$LIBRARY: BASIC$STARLET"
%INCLUDE "$SSDEF" %FROM %LIBRARY "SYS$LIBRARY: BASIC$STARLET"

COMMON LONG NUM_COMMANDS

DECLARE LONG S

EXTERNAL LONG FUNCTION LIB$SIGNAL(LONG), &
    SMG$PUT_LINE (LONG, STRING)

NUM_COMMANDS = NUM_COMMANDS - 1
```

SMG\$CREATE_SUBPROCESS

```
IF (SMG_INFO::SMG$L_STATUS AND 1) <> 0
THEN
  S = SMG$PUT_LINE(SMG_INFO::SMG$L_USR_ARG, "command completed")
  IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL (S) END IF
ELSE
  S = SMG$PUT_LINE(SMG_INFO::SMG$L_USR_ARG, "command failed")
  IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL (S) END IF
END IF
SUBEND
```

SMG\$CREATE_VIEWPORT

SMG\$CREATE_VIEWPORT—Create a Virtual Viewport

The Create a Virtual Viewport routine creates a viewport and associates it with a virtual display. The location and size of the viewport are specified by the caller.

Format

```
SMG$CREATE_VIEWPORT  display-id ,viewport-row-start ,viewport-column-start ,viewport-number-rows  
                    ,viewport-number-columns
```

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

display-id

OpenVMS usage	identifier
type	longword (unsigned)
access	read only
mechanism	by reference

Display identifier of the virtual display associated with the newly created viewport. The **display-id** argument is the address of an unsigned longword containing this identifier.

viewport-row-start

OpenVMS usage	longword_signed
type	longword (signed)
access	read only
mechanism	by reference

Row number in the virtual display that will become row 1 in the viewport. The **viewport-row-start** argument is the address of a signed longword containing the row number.

viewport-column-start

OpenVMS usage longword_signed
 type longword (signed)
 access read only
 mechanism by reference

Column number in the virtual display that will become column 1 in the viewport. The **viewport-column-start** argument is the address of a signed longword containing the column number.

In the case that the **viewport-column-start** begins on the right portion of a Chinese character, the right portion of the character becomes invisible.

viewport-number-rows

OpenVMS usage longword_signed
 type longword (signed)
 access read only
 mechanism by reference

Number of rows in the viewport. The **viewport-number-rows** argument is the address of a signed longword containing the number of rows in the newly created viewport.

viewport-number-columns

OpenVMS usage longword_signed
 type longword (signed)
 access read only
 mechanism by reference

Number of columns in the viewport. The **viewport-number-columns** argument is the address of a signed longword containing the number of columns in the newly created viewport.

Description

SMG\$CREATE_VIEWPORT creates a viewport and associates it with a particular virtual display. The virtual display must be created before the viewport can be created, and you can only create one viewport for each virtual display. In order to make the viewport visible, you have to paste the virtual display by calling the SMG\$PASTE_VIRTUAL_DISPLAY routine; only the portion of the virtual display that falls inside the viewport is visible. You can delete a viewport with the SMG\$DELETE_VIEWPORT routine.

In case Chinese language character set is used, and the **viewport-column-start** begins on the right portion of a Chinese character, the right portion of the character will become invisible. Also if the viewport ends on the left portion of a Chinese character, the left portion of the character will become invisible.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
SMG\$_INVARG	Number of rows or columns is less than zero.
SMG\$_INVCOL	Invalid column specified.
SMG\$_INVDIS_ID	Invalid display-id .

SMG\$CREATE_VIEWPORT

SMG\$_INVROW	Invalid row specified.
SMG\$_WINEXISTS	Viewport already exists on the virtual display (alternate success status).
SMG\$_WRONUMARG	Wrong number of arguments.

Example

```
C+
C This DEC Fortran example creates two virtual displays, one
C being a copy of the other. The initial virtual display is
C filled and pasted to the pasteboard. The second virtual
C display is assigned a viewport and then pasted to the
C pasteboard. Therefore, only the section of the second
C virtual display that falls inside the viewport is visible.
C-
      IMPLICIT INTEGER (A-Z)
      INCLUDE '($SMGDEF)'

C Create the Virtual Display. Give it a border.
      ROWS = 9
      COLUMNS = 32
      STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1      (ROWS, COLUMNS, DISPLAY1, SMG$_M_BORDER )
      IF (.NOT. STATUS) call lib$signal(%val(STATUS))

C Create the Pasteboard
      STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
      IF (.NOT. STATUS) call lib$signal(%val(STATUS))

C Put data in the Virtual Display
      STATUS = SMG$PUT_CHARS ( DISPLAY1,
1      'This is row number 1, you see.', 1, 1)
      IF (.not. STATUS) call lib$signal(%val(STATUS))
      STATUS = SMG$PUT_CHARS ( DISPLAY1,
1      'This is row number 2, you see.', 2, 1)
      IF (.not. STATUS) call lib$signal(%val(STATUS))
      STATUS = SMG$PUT_CHARS ( DISPLAY1,
1      'This is row number 3, you see.', 3, 1)
      IF (.not. STATUS) call lib$signal(%val(STATUS))
      STATUS = SMG$PUT_CHARS ( DISPLAY1,
1      'This is row number 4, you see.', 4, 1)
      IF (.not. STATUS) call lib$signal(%val(STATUS))
      STATUS = SMG$PUT_CHARS ( DISPLAY1,
1      'This is row number 5, you see.', 5, 1)
      IF (.not. STATUS) call lib$signal(%val(STATUS))
      STATUS = SMG$PUT_CHARS ( DISPLAY1,
1      'This is row number 6, you see.', 6, 1)
      IF (.not. STATUS) call lib$signal(%val(STATUS))
      STATUS = SMG$PUT_CHARS ( DISPLAY1,
1      'This is row number 7, you see.', 7, 1)
      IF (.not. STATUS) call lib$signal(%val(STATUS))
      STATUS = SMG$PUT_CHARS ( DISPLAY1,
1      'This is row number 8, you see.', 8, 1)
      IF (.not. STATUS) call lib$signal(%val(STATUS))
      STATUS = SMG$PUT_CHARS ( DISPLAY1,
1      'This is row number 9, you see.', 9, 1)
      IF (.not. STATUS) call lib$signal(%val(STATUS))
```

SMG\$CREATE_VIEWPORT

```
C Paste the Virtual Display
STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 2, 2)
IF (.NOT. STATUS) call lib$signal(%VAL(STATUS))

STATUS = SMG$LABEL_BORDER (DISPLAY1, 'Full Display',,,SMG$M_BOLD)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$COPY_VIRTUAL_DISPLAY (DISPLAY1, DISPLAY2)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$LABEL_BORDER (DISPLAY2, 'Viewport',,,SMG$M_BOLD)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$CREATE_VIEWPORT ( DISPLAY2, 3, 9, 3, 12)
IF (.NOT. STATUS) call lib$signal(%VAL(STATUS))

STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DISPLAY2, PASTE1, 15, 20)
IF (.NOT. STATUS) call lib$signal(%VAL(STATUS))

END
```

In this DEC Fortran example, the initial virtual display is copied to a second virtual display that has a viewport associated with it. When the second virtual display is pasted, only the portion of the virtual display that falls inside the viewport is visible. This is shown in Figure SMG-8.

Figure SMG-8 Output Generated by Creating a Viewport

```
Full Display
This is row number 1, you see.
This is row number 2, you see.
This is row number 3, you see.
This is row number 4, you see.
This is row number 5, you see.
This is row number 6, you see.
This is row number 7, you see.
This is row number 8, you see.
This is row number 9, you see.

Viewport
row number 3
row number 4
row number 5
```

ZK-6424-GE

SMG\$CREATE_VIRTUAL_DISPLAY

SMG\$CREATE_VIRTUAL_DISPLAY—Create a Virtual Display

The Create Virtual Display routine creates a virtual display and returns its assigned display identifier.

Format

```
SMG$CREATE_VIRTUAL_DISPLAY  number-of-rows ,number-of-columns ,display-id [,display-attributes]
                             [,video-attributes] [,character-set]
```

Returns

OpenVMS usage	cond_value
type	longword (unsigned)
access	write only
mechanism	by value

Arguments

number-of-rows

OpenVMS usage	longword_signed
type	longword (signed)
access	read only
mechanism	by reference

Specifies the number of rows in the newly created virtual display. The **number-of-rows** argument is the address of a signed longword that contains the desired number of rows.

number-of-columns

OpenVMS usage	longword_signed
type	longword (signed)
access	read only
mechanism	by reference

Specifies the number of columns in the newly created virtual display. The **number-of-columns** argument is the address of a signed longword that contains the desired number of columns.

display-id

OpenVMS usage	identifier
type	longword (unsigned)
access	write only
mechanism	by reference

Receives the **display-id** of the newly created virtual display. The **display-id** argument is the address of an unsigned longword into which is written the display identifier.

display-attributes

OpenVMS usage	mask_longword
type	longword (unsigned)
access	read only
mechanism	by reference

SMG\$CREATE_VIRTUAL_DISPLAY

Specifies the default display attributes. The optional **display-attributes** argument is the address of an unsigned longword that contains the desired display attributes.

Valid values for **display-attributes** are as follows:

SMG\$M_BORDER	Specifies a bordered display. If omitted, the display is not bordered.
SMG\$M_BLOCK_BORDER	Specifies a block-bordered display. If omitted, the display is not bordered.
SMG\$M_DISPLAY_CONTROLS	Specifies that control characters such as carriage return and line feed are displayed as graphic characters, if your terminal supports them.
SMG\$M_TRUNC_ICON	Specifies that an icon (generally a diamond shape) is displayed where truncation of a line exceeding the width of the virtual display has occurred.

video-attributes

OpenVMS usage	mask_longword
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the default rendition to be applied to all output in this virtual display unless overridden by a call to a specific output routine (for example, SMG\$CHANGE_RENDITION). The **video-attributes** argument is the address of an unsigned longword that contains the video attributes mask.

Valid values for this argument are as follows:

SMG\$M_BLINK	Displays blinking characters.
SMG\$M_BOLD	Displays characters in higher-than-normal intensity.
SMG\$M_REVERSE	Displays characters in reverse video, that is, using the opposite of the default rendition of the virtual display.
SMG\$M_UNDERLINE	Displays underlined characters.
SMG\$M_INVISIBLE	Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard.
SMG\$M_USER1 through SMG\$M_USER8	Displays user-defined attributes.

character-set

OpenVMS usage	longword_unsigned
type	longword (unsigned)
access	read only
mechanism	by reference

Specifies the default character set for all text in this virtual display. The **character-set** argument is the address of an unsigned longword that contains the character set specifier. If this argument is omitted, the logical name of SMG\$DEFAULT_CHARACTER_SET is used to determine the default character set. If the logical name is not defined or if the logical name is invalid, the default character set value is UNKNOWN.

SMG\$CREATE_VIRTUAL_DISPLAY

The valid values for this argument is as follows:

Value	Character Set
SMG\$C_ASCII	ASCII
SMG\$C_SPEC_GRAPHICS	DEC Special Graphics
SMG\$C_HANYU	DEC Hanyu

For the valid SMG\$DEFAULT_CHARACTER_SET definitions, refer to Table 2-1 in Section 2.2.14.

Description

SMG\$CREATE_VIRTUAL_DISPLAY creates a new virtual display and returns its display identifier. Initially, the virtual display contains blanks, and the virtual cursor is positioned at row 1, column 1. The virtual scrolling region is the entire virtual display. To make the display visible, use the SMG\$PASTE_VIRTUAL_DISPLAY routine.

Original SMG routines support only ASCII(default) and DEC Special Graphics, but Chinese SMG will allow users to specify the **character-set** which enables Chinese language to be used.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
LIB\$_INSVIRMEM	Insufficient virtual memory.
SMG\$_INVARG	Invalid argument. Video-attributes or display-attributes contains an unknown value.
SMG\$_WRONUMARG	Wrong number of arguments.

Examples

- ```
C+
C This DEC Fortran example program demonstrates the use of
C SMG$ERASE_PASTEBOARD.
C-

 IMPLICIT INTEGER*4 (A-Z)
 CHARACTER*80 OUT_STR,TRIM_STR
 CHARACTER*18 PROMPT //'Please enter data '/

 SMG$M_BOLD = 1
 SMG$M_REVERSE = 2
 SMG$M_BLINK = 4
 SMG$M_UNDERLINE = 8

C+
C Establish the terminal keyboard as the virtual keyboard
C by calling SMG$CREATE_VIRTUAL_KEYBOARD.
C-

 STATUS = SMG$CREATE_VIRTUAL_KEYBOARD(KEYBOARD_ID,,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))

C+
C Establish the terminal screen as a pasteboard using
C SMG$CREATE_PASTEBOARD.
C-
```

## SMG\$CREATE\_VIRTUAL\_DISPLAY

```
 STATUS = SMG$CREATE_PASTEBOARD (NEW_PID,,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Establish a virtual display region by
C calling SMG$CREATE_VIRTUAL_DISPLAY.
C-
 STATUS = SMG$CREATE_VIRTUAL_DISPLAY (5,80,DISPLAY_ID,,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Paste the virtual display to the screen, starting at
C row 10, column 15. To paste the virtual display, use
C SMG$PASTE_VIRTUAL_DISPLAY.
C-
 STATUS = SMG$PASTE_VIRTUAL_DISPLAY(DISPLAY_ID,NEW_PID,10,15)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Prompt the user for input, and accept that input using
C SMG$READ_STRING.
C-
 STATUS = SMG$READ_STRING(KEYBOARD_ID,OUT_STR,PROMPT,,,,,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Clear the screen using SMG$ERASE_PASTEBOARD.
C-
 STATUS = SMG$ERASE_PASTEBOARD (NEW_PID)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Trim any trailing blanks from the user input
C by calling STR$TRIM.
C-
 STATUS = STR$TRIM(TRIM_STR,OUT_STR,STR_LEN)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Display the data input by the user using SMG$PUT_CHARS
C and SMG$PUT_LINE.
C-
 STATUS = SMG$PUT_CHARS(DISPLAY_ID,'You entered: ',,,,,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
 STATUS = SMG$PUT_LINE(DISPLAY_ID,TRIM_STR(1:STR_LEN),,
1 SMG$M_REVERSE,0,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
 END
```

The output generated by this FORTRAN example is shown in Figure SMG-9.

## SMG\$CREATE\_VIRTUAL\_DISPLAY

Figure SMG-9 Output of FORTRAN Program Calling SMG\$CREATE\_VIRTUAL\_DISPLAY



ZK-4100-GE

---

## SMG\$CREATE\_VIRTUAL\_KEYBOARD—Create a Virtual Keyboard

The Create Virtual Keyboard routine creates a virtual keyboard and returns its assigned keyboard identifier.

### Format

```
SMG$CREATE_VIRTUAL_KEYBOARD keyboard-id [,input-device] [,default-filespec] [,resultant-filespec]
 [,recall-size]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### keyboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Receives the keyboard identifier of the newly created virtual keyboard. The **keyboard-id** argument is the address of an unsigned longword into which is written the keyboard identifier.

#### input-device

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String containing the file specification or logical name of the file or terminal to be used for this virtual keyboard. The **input-device** argument is the address of a descriptor pointing to the file specification. If omitted, this defaults to SYS\$INPUT.

---

#### Note

---

An RMS file is not supported as an input device in Chinese Screen Management Facility.

---

#### default-filespec

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String containing the default file specification. The **default-filespec** argument is the address of a descriptor pointing to the default file specification. If omitted, the null string is used.

## SMG\$CREATE\_VIRTUAL\_KEYBOARD

**Default-filespec** might be used to specify a default device and directory, leaving the **input-device** argument to supply the file name and type.

---

### Note

---

An RMS file is not supported as an input device in Chinese Screen Management Facility.

---

### resultant-filespec

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

String into which the procedure writes the fully expanded file specification of the file used. The **resultant-filespec** argument is the address of a descriptor pointing to the string into which is written the file specification that was used.

---

### Note

---

An RMS file is not supported as an input device in Chinese Screen Management Facility.

---

### recall-size

|               |                 |
|---------------|-----------------|
| OpenVMS usage | byte_unsigned   |
| type          | byte (unsigned) |
| access        | read only       |
| mechanism     | by reference    |

Number of input lines to be saved for later recall. The optional **recall-size** argument is the address of an unsigned byte containing the specified number of lines. A value of zero turns off input line recall. By default, 20 lines are saved for later recall.

## Description

SMG\$CREATE\_VIRTUAL\_KEYBOARD creates the association between a terminal name and a virtual keyboard. The keyboard identifier is then passed to other SMG\$ procedures in order to identify the input stream being acted upon.

If your program also calls the routine SMG\$CREATE\_PASTEBOARD, be sure to call that routine *before* you call SMG\$CREATE\_VIRTUAL\_KEYBOARD.

This procedure assigns a channel to the terminal and sets the terminal's keyboard to application mode (if supported). These attributes are restored to their previous values when the virtual keyboard is deleted. The virtual keyboard is deleted automatically when the image exits and can also be deleted by a call to SMG\$DELETE\_VIRTUAL\_KEYBOARD.

## Condition Values Returned

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                         |
| SMG\$_FILTOOLON | File specification is too long (over 255 characters). |
| SMG\$_WRONUMARG | Wrong number of arguments.                            |
| LIB\$_INSEF     | Insufficient number of event flags.                   |
| LIB\$_INSVIRMEM | Insufficient virtual memory.                          |
| LIB\$_INVSTRDES | Invalid string descriptor.                            |

Any condition values returned by \$GETDVIW, \$ASSIGN, or \$DCLEXH.

## Example

```

C+
C This DEC Fortran example program demonstrates the use of
C SMG$CREATE_VIRTUAL_KEYBOARD, SMG$CREATE_KEY_TABLE,
C SMG$ADD_KEY_DEF, and SMG$READ_COMPOSED_LINE.
C-

 INTEGER SMG$CREATE_VIRTUAL_KEYBOARD, SMG$CREATE_KEY_TABLE
 INTEGER SMGADD_KEY_DEF, SMGREAD_COMPOSED_LINE
 INTEGER SMG$DELETE_KEY_DEF, KEYBOARD, KEYTABLE, STATUS

C+
C Include the SMG definitions. In particular, we want SMG$M_KEY_NOECHO
C and SMG$M_KEY_TERMINATE.
C-

 INCLUDE '($SMGDEF)'

C+
C Create a virtual keyboard (using SMG$CREATE_VIRTUAL_KEYBOARD)
C and create a key table (using SMG$CREATE_KEY_TABLE).
C-

 STATUS = SMG$CREATE_VIRTUAL_KEYBOARD (KEYBOARD)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$CREATE_KEY_TABLE (KEYTABLE)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Prompt the user with the following instructions.
C-

 WRITE (6,*) 'When you see the prompt (->), strike the following'
 WRITE (6,*) 'keys (on the KEYPAD): '
 WRITE (6,*) ' PF1 '
 WRITE (6,*) ' 5 '
 WRITE (6,*) ' PF3 '
 WRITE (6,*) ' '
 WRITE (6,*) 'When you have done this, the following sentence'
 WRITE (6,*) '(and nothing more) should appear following the'
 WRITE (6,*) 'prompt: '
 WRITE (6,*) '(PF3 should act as a carriage return.)'
 WRITE (6,*) ' '
 WRITE (6,*) 'NOW IS THE TIME FOR ALL TEXT TO APPEAR.'

C+
C Add key definitions by calling SMG$ADD_KEY_DEF.
C-

```

## SMG\$CREATE\_VIRTUAL\_KEYBOARD

```
STATUS = SMG$ADD_KEY_DEF (KEYTABLE, 'PF1', , ,
1 'NOW IS THE TIME FOR ')
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$ADD_KEY_DEF (KEYTABLE, 'KP5', , ,
1 'TEXT TO APPEAR.')
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$ADD_KEY_DEF (KEYTABLE, 'PF3', , ,
1 SMG$M_KEY_NOECHO + SMG$M_KEY_TERMINATE ,
1 'THIS SHOULD NOT BE ECHOED. IF YOU CAN
1 SEE THIS, AN ERROR EXISTS.')
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$READ_COMPOSED_LINE to read a line of input.
C-

WRITE(6,*) ' '
STATUS = SMG$READ_COMPOSED_LINE (KEYBOARD, KEYTABLE, R_TEXT,
1 '->')
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

END
```

### Output session:

```
$ RUN example
```

When you see the prompt (->), strike the following keys (on the KEYPAD):

```
PF1
5
PF3
```

When you have done this, the following sentence (and nothing more) should appear following the prompt:  
(PF3 should act as a carriage return.)

```
NOW IS THE TIME FOR ALL TEXT TO APPEAR.
->NOW IS THE TIME FOR ALL TEXT TO APPEAR.
$
```



---

## SMG\$CURSOR\_COLUMN—Return Cursor Column Position

The Return Cursor Column Position routine returns the virtual cursor's current column position in a specified virtual display.

### Format

SMG\$CURSOR\_COLUMN display-id

### Returns

OpenVMS usage longword (unsigned)  
 type write only  
 access by value

SMG\$CURSOR\_COLUMN returns the current virtual cursor column position.

### Arguments

**display-id**  
 OpenVMS usage identifier  
 type longword (unsigned)  
 access read only  
 mechanism by reference

The display for which the column position is returned. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

### Description

SMG\$CURSOR\_COLUMN returns a longword containing the value of the current virtual cursor column position for the specified virtual display. If the **display-id** is omitted, this routine signals SMG\$\_WRONUMARG. If the **display-id** is invalid, this routine signals SMG\$\_INVDIS\_ID.

### Condition Values Signaled

|                 |                             |
|-----------------|-----------------------------|
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> . |
| SMG\$_WRONUMARG | Wrong number of arguments.  |

## SMG\$CURSOR\_ROW

---

### SMG\$CURSOR\_ROW—Return Cursor Row Position

The Return Cursor Row Position routine returns the virtual cursor's current row position in a specified virtual display.

#### Format

SMG\$CURSOR\_ROW display-id

#### Returns

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access write only  
mechanism by value

SMG\$CURSOR\_ROW returns the current row position.

#### Arguments

**display-id**  
OpenVMS usage identifier  
type longword (unsigned)  
access read only  
mechanism by reference

The display for which the row position is returned. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### Description

SMG\$CURSOR\_ROW returns a longword containing the value of the current virtual cursor row position for the specified virtual display. If the **display-id** is omitted, this routine signals SMG\$\_WRONUMARG. If the **display-id** is invalid, this routine signals SMG\$\_INVDIS\_ID.

#### Condition Values Signaled

|                 |                             |
|-----------------|-----------------------------|
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> . |
| SMG\$_WRONUMARG | Wrong number of arguments.  |

---

## SMG\$DEFINE\_KEY—Perform a DEFINE/KEY Command

The Perform a DEFINE/KEY Command routine performs the DEFINE/KEY command you provide.

### Format

```
SMG$DEFINE_KEY key-table-id ,command-string
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### **key-table-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identification of the key definition table for which the DEFINE/KEY command is to be performed. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

**Key-table-id** is returned by SMG\$CREATE\_KEY\_TABLE.

#### **command-string**

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String containing the DEFINE/KEY command to be performed. The **command-string** argument is the address of a descriptor pointing to the command to be performed.

The valid qualifiers for the DEFINE/KEY command are as follows:

- /TERMINATE
- /NOECHO
- /LOCK
- /IF\_STATE
- /SET\_STATE

The following two restrictions apply to the DEFINE/KEY qualifiers:

- If you use the /LOCK qualifier, you must also use the /SET\_STATE qualifier.
- If you use both the /SET\_STATE and /TERMINATE qualifiers, you may not use /LOCK.

## SMG\$DEFINE\_KEY

### Description

SMG\$DEFINE\_KEY parses and performs a DEFINE/KEY command. It can be used by programs that accept DEFINE/KEY commands but do not parse the commands themselves.

SMG\$DEFINE\_KEY calls CLI\$DCL\_PARSE to parse the command line and then makes the appropriate call to SMG\$ADD\_KEY\_DEF. The original command is then restored with a call to CLI\$DCL\_PARSE. Use of this procedure requires that the image be run under the DCL Command Language Interpreter.

### Condition Values Returned

SS\$\_NORMAL                      Normal successful completion.

SMG\$\_WRONUMARG                Wrong number of arguments.

Any condition values returned by LIB\$SCOPY\_DXDX.

Any condition values returned by CLI\$ routines.

Any condition values returned by SMG\$ADD\_KEY\_DEF.

---

## SMG\$DEL\_TERM\_TABLE—Delete Terminal Table

The Delete Terminal Table routine terminates access to a private TERMTABLE.EXE and frees the associated virtual address space.

### Format

SMG\$DEL\_TERM\_TABLE

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

None.

### Description

SMG\$DEL\_TERM\_TABLE terminates access to a private TERMTABLE.EXE. Calling this routine is optional. This routine is useful in the case where a calling program might need to reuse the virtual address space used by a private TERMTABLE. This routine should be used only when you perform direct (non-SMG\$) I/O to terminals.

### Condition Values Returned

|            |                               |
|------------|-------------------------------|
| SS\$NORMAL | Normal successful completion. |
|------------|-------------------------------|

## SMG\$DELETE\_CHARS

---

### SMG\$DELETE\_CHARS—Delete Characters

The Delete Characters routine deletes columns in a virtual display.

#### Format

SMG\$DELETE\_CHARS display-id ,number-of-columns ,start-row ,start-column

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifies the virtual display from which columns are to be deleted. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### number-of-columns

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the number of columns to be deleted. The **number-of-columns** argument is the address of a signed longword that contains the number of columns to be deleted.

##### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row position at which to start the deletion. The **start-row** argument is the address of a signed longword that contains the row number at which to start the deletion.

##### start-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column position at which to start the deletion. The **start-column** argument is the address of a signed longword that contains the column position at which to start the deletion.

In case that the start-column starts on the right portion of a Chinese character, the left portion of the character will become an undefined character.

## Description

SMG\$DELETE\_CHARS deletes a specified number of columns, starting at a specified row and column position. Remaining columns on the line are shifted to the left to occupy the vacated space(s). Note that this routine deletes columns only on a single line.

If you specify more columns than are available for deletion, SMG\$DELETE\_CHARS deletes all columns from the specified column position to the end of the line.

In case Chinese language character set is used, and the start column begins on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if the deletion terminates on the left portion of a Chinese character, the right portion of the character will become an undefined character.

This routine leaves the virtual cursor at the position of the first columns deleted.

## Condition Values Returned

|                 |                                                                                        |
|-----------------|----------------------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                                          |
| SMG\$_INVARG    | Invalid argument. The number of columns specified extends outside the virtual display. |
| SMG\$_INVCOL    | Invalid column position. The specified column is outside the virtual display.          |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                                            |
| SMG\$_INVROW    | Invalid row position. The specified row is outside the virtual display.                |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal.                                                    |
| SMG\$_WRONUMARG | Wrong number of arguments.                                                             |

## Example

```
C+
C This DEC Fortran example program demonstrates the use of
C SMG$DELETE_CHARS.
C-
 INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
 INTEGER SMG$PASTE_VIRTUAL_DISPLAY, SMG$PUT_CHARS
 INTEGER SMG$DELETE_CHARS, DISPLAY1, PASTE1
 INTEGER ROWS, COLUMNS, BORDER, STATUS
C+
C Create the virtual display by calling SMG$CREATE_VIRTUAL_DISPLAY.
C To give it a border, set BORDER = 1. No border would be BORDER = 0.
C-
 ROWS = 7
 COLUMNS = 50
 BORDER = 1
```

## SMG\$DELETE\_CHARS

```
 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PUT_CHARS to put data in the virtual display.
C-

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This virtual display has 7 rows and 50 columns.', 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This is a bordered virtual display.', 4, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Paste the virtual display to the pasteboard using
C SMG$PASTE_VIRTUAL_DISPLAY.
C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$DELETE_CHARS to delete 4 characters from row 4
C starting from character (column) 14, removing the characters
C "rder" from the word "bordered".
C-

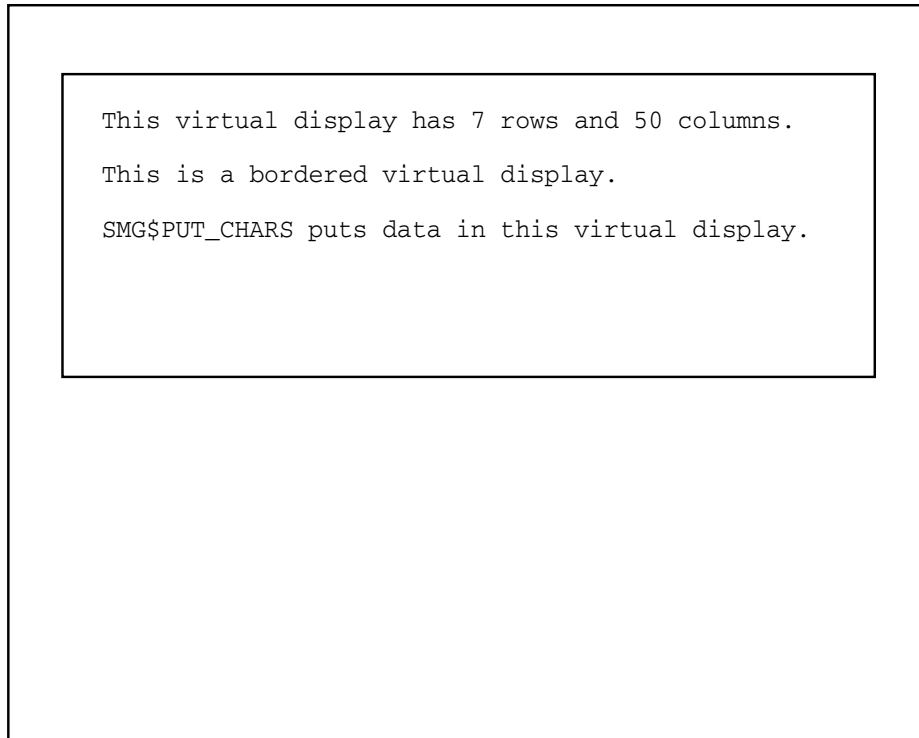
 STATUS = SMG$DELETE_CHARS (DISPLAY1, 4, 4, 14)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 END
```

The output generated by this FORTRAN program before the call to SMG\$DELETE\_CHARS is shown in Figure SMG-10.



**Figure SMG-10 Output Generated Before the Call to SMG\$DELETE\_CHARS**

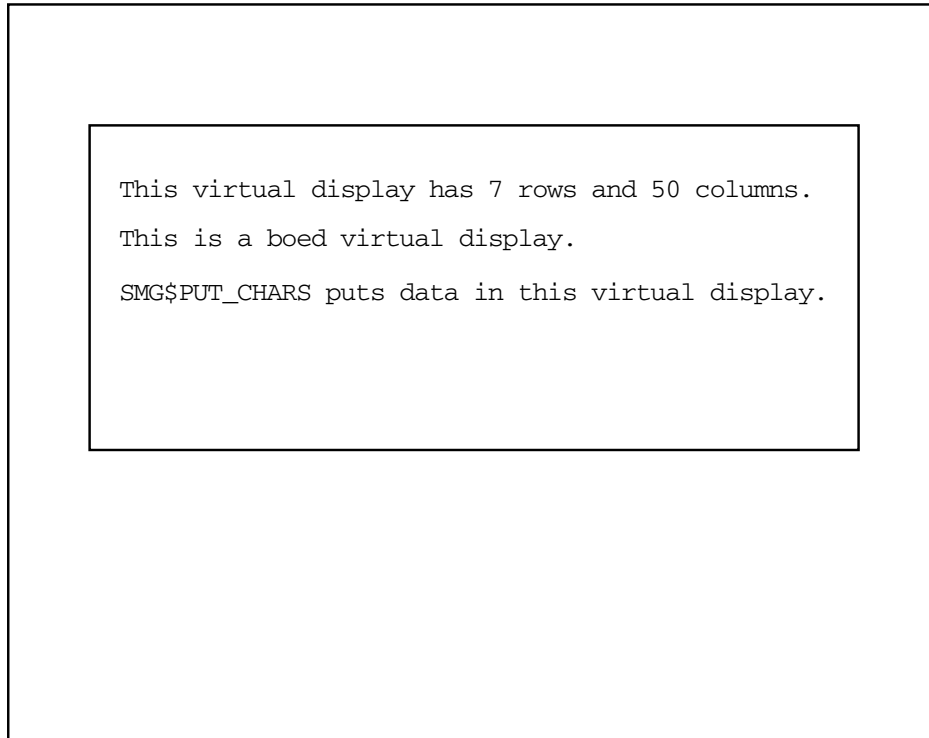


ZK-4101-GE

The output generated after the call to SMG\$DELETE\_CHARS is shown in Figure SMG-11.

## SMG\$DELETE\_CHARS

Figure SMG-11 Output Generated After the Call to SMG\$DELETE\_CHARS



ZK-4107-GE

---

## SMG\$DELETE\_KEY\_DEF—Delete Key Definition

The Delete Key Definition routine deletes a key definition from the specified table of key definitions.

### Format

```
SMG$DELETE_KEY_DEF key-table-id ,key-name [,if-state]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### key-table-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifies the key table from which the key definition is deleted. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

#### key-name

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String containing the name of the key whose definition is to be deleted. The **key-name** argument is the address of a descriptor pointing to the key name. **Key-name** is stripped of trailing blanks and converted to uppercase before use.

Table 3-1 in Part I of this manual lists the valid key names.

## SMG\$DELETE\_KEY\_DEF

### **if-state**

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String containing a state name which further qualifies **key-name**. The **if-state** argument is the address of a descriptor pointing to the state name. If omitted, the null state is used. Thus if a key has several definitions depending on various values of **if-state**, this routine lets you delete only one of those definitions.

### **Description**

SMG\$DELETE\_KEY\_DEF deletes a key definition from the specified table of key definitions.

### **Condition Values Returned**

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVKEYNAM | Invalid <b>key-name</b> .     |
| SMG\$_INVKTB_ID | Invalid <b>key-table-id</b> . |
| SMG\$_KEYNOTDEF | Key is not defined.           |
| SMG\$_WRONUMARG | Wrong number of arguments.    |
| SMG\$_KEYDEFPRO | Key definition is protected.  |

---

## SMG\$DELETE\_LINE—Delete Line

The Delete Line routine deletes lines from a virtual display.

### Format

SMG\$DELETE\_LINE display-id ,start-row [,number-of-rows]

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifies the virtual display from which lines are to be deleted. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the first line to be deleted from the virtual display. The **start-row** argument is the address of a signed longword that contains the number of the first line to be deleted.

#### number-of-rows

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the number of lines to be deleted. The **number-of-rows** argument is the address of a signed longword that contains the number of lines to be deleted. If omitted, one line is deleted.

### Description

SMG\$DELETE\_LINE deletes one or more lines from a virtual display and scrolls the remaining lines up into the space created by the deletion. Blank lines fill the display on the bottom. The virtual cursor is left at the first column position in **start-row**.

## SMG\$DELETE\_LINE

### Condition Values Returned

|                 |                                     |
|-----------------|-------------------------------------|
| SS\$_NORMAL     | Normal successful completion.       |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .         |
| SMG\$_WRONUMARG | Wrong number of arguments.          |
| SMG\$_INVROW    | Invalid row.                        |
| SMG\$_INVARG    | Invalid argument.                   |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal. |

### Example

```
C+
C This DEC Fortran example program demonstrates the use of SMG$DELETE_LINE.
C-

 IMPLICIT INTEGER (A-Z)

C+
C Create the virtual display by calling SMG$CREATE_VIRTUAL_DISPLAY.
C To give it a border, set BORDER = 1. No border would be BORDER = 0.
C-

 ROWS = 7
 COLUMNS = 50
 BORDER = 1

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PUT_CHARS to put data in the virtual display.
C-

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This virtual display has 7 rows and 50 columns.', 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This is a bordered virtual display.', 4, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Paste the virtual display to the pasteboard using
C SMG$PASTE_VIRTUAL_DISPLAY.
C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

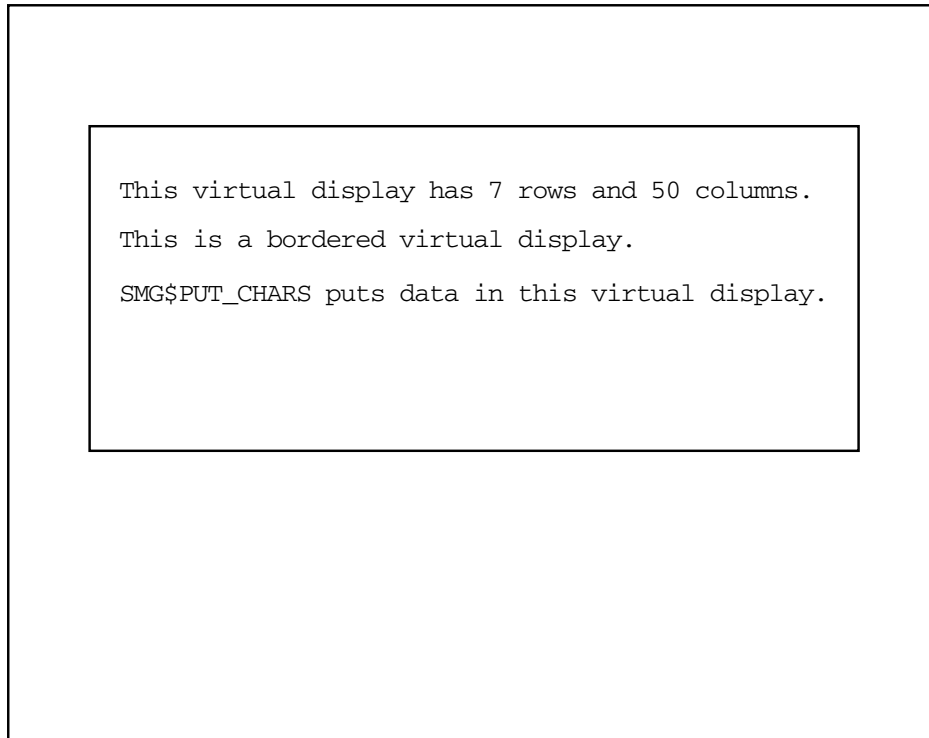
C+
C Call SMG$DELETE_LINE to delete rows 3, 4, and 5.
C-

 STATUS = SMG$DELETE_LINE (DISPLAY1, 3, 3)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

END

The output generated by this FORTRAN program before the call to SMG\$DELETE\_LINE is shown in Figure SMG-12.

**Figure SMG-12 Output Generated by FORTRAN Program Before the Call to SMG\$DELETE\_LINE**

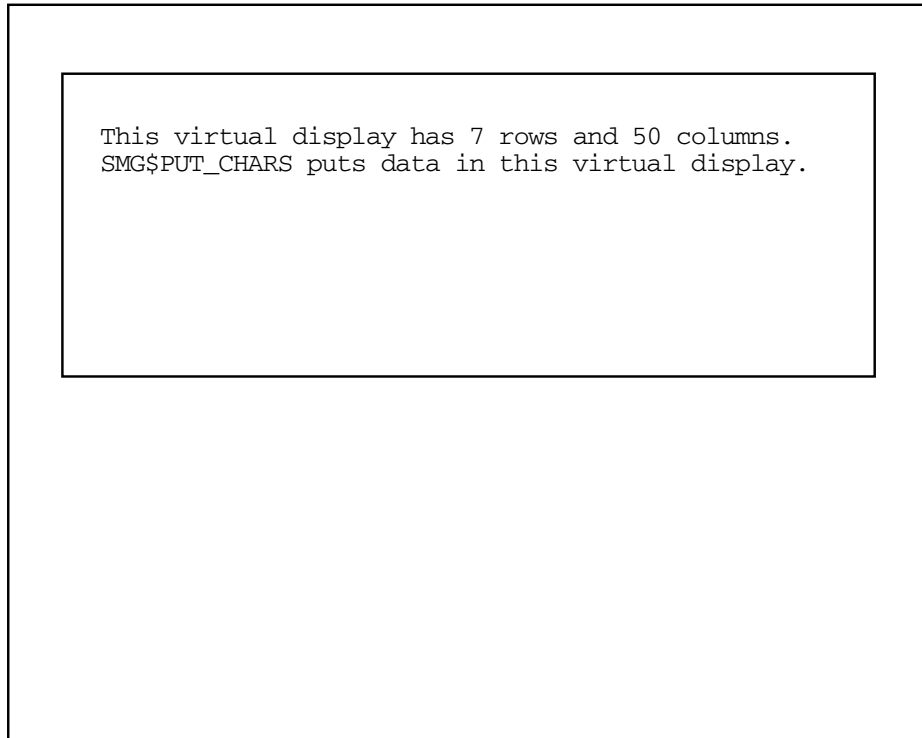


ZK-4103-GE

The output generated after the call to SMG\$DELETE\_LINE is shown in Figure SMG-13.

## SMG\$DELETE\_LINE

Figure SMG-13 Output Generated After the Call to SMG\$DELETE\_LINE



ZK-4109-GE



---

## SMG\$DELETE\_MENU—End Access to a Menu in the Virtual Display

The End Access to a Menu in the Virtual Display routine ends access to the menu choices in the specified virtual display.

### Format

SMG\$DELETE\_MENU display-id [,flags]

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the virtual display in which the menu choices are displayed. The **display-id** argument is the address of an unsigned longword containing this identifier.

#### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask specifying possible actions to take when deleting the menu. The **flags** argument is the address of an unsigned longword that contains the flag. At this time, the only valid value is SMG\$M\_ERASE\_MENU. If this option is specified, all rows containing menu items are erased.

### Description

SMG\$DELETE\_MENU discontinues access to the menu choices in the specified virtual display. The optional **flags** argument lets you specify that the menu choices be removed from the display when the menu is deleted.

### Condition Values Returned

|             |                                                     |
|-------------|-----------------------------------------------------|
| SS\$_NORMAL | Normal successful completion.                       |
| LIB\$_xxxx  | Any condition value returned by LIB\$FREE_VM.       |
| SMG\$_xxxx  | Any condition value returned by SMG\$ERASE_DISPLAY. |

## SMG\$DELETE\_PASTEBOARD

---

### SMG\$DELETE\_PASTEBOARD—Delete Pasteboard

The Delete Pasteboard routine deletes a pasteboard.

#### Format

SMG\$DELETE\_PASTEBOARD pasteboard-id [,flags]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **pasteboard-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard to be deleted. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

##### **flags**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask specifying whether the screen is cleared after the specified pasteboard is deleted. The **flags** argument is the address of an unsigned longword that contains the flag. Valid values are as follows:

|                  |                              |
|------------------|------------------------------|
| 0                | Does not clear the screen.   |
| SMG\$M_ERASE_PBD | Clears the screen (default). |

#### Description

SMG\$DELETE\_PASTEBOARD flushes all output to the display, terminates all use of the specified pasteboard, and deallocates all resources associated with the pasteboard.

#### Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_WRONUMARG | Wrong number of arguments.     |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |

## SMG\$DELETE\_PASTEBOARD

SMG\$\_NOTPASTED            The specified virtual display is not pasted to the specified pasteboard.

SMG\$\_WILUSERMS           Pasteboard is not a video terminal.

Any condition values returned by \$DASSGN, LIB\$FREE\_VM, LIB\$FREE\_EF, or SMG\$FLUSH\_BUFFER.

## SMG\$DELETE\_SUBPROCESS

---

### SMG\$DELETE\_SUBPROCESS—Terminate a Subprocess

The Terminate a Subprocess routine deletes a subprocess that was created with the SMG\$CREATE\_SUBPROCESS routine.

#### Format

SMG\$DELETE\_SUBPROCESS display-id

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

|                   |                     |
|-------------------|---------------------|
| <b>display-id</b> |                     |
| OpenVMS usage     | identifier          |
| type              | longword (unsigned) |
| access            | read only           |
| mechanism         | by reference        |

Identifier of the virtual display associated with the subprocess being deleted. The **display-id** argument is the address of an unsigned longword that contains this virtual display identifier.

#### Description

SMG\$DELETE\_SUBPROCESS deletes a subprocess that was created by a call to SMG\$CREATE\_SUBPROCESS. Because the Screen Management Facility provides its own exit handlers, do not invoke SMG\$DELETE\_SUBPROCESS from within your own exit handler. For more information, see Section 4.4 in Chapter 4.

#### Condition Values Returned

|                 |                                      |
|-----------------|--------------------------------------|
| SS\$_NORMAL     | Normal successful completion.        |
| SMG\$_NOSUBEXI  | No subprocess exists.                |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .          |
| LIB\$_xxxx      | Any status returned by LIB\$FREE_VM. |
| SS\$_xxxx       | Any status returned by \$DELPRC.     |

---

## SMG\$DELETE\_VIEWPORT—Delete a Viewport

The Delete a Viewport routine deletes the specified viewport from any pasteboards to which it is pasted.

### Format

SMG\$DELETE\_VIEWPORT display-id

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

|                   |                     |
|-------------------|---------------------|
| <b>display-id</b> |                     |
| OpenVMS usage     | identifier          |
| type              | longword (unsigned) |
| access            | read only           |
| mechanism         | by reference        |

Identifier of the virtual display associated with the viewport to be deleted. The **display-id** argument is the address of an unsigned longword containing the display identifier.

### Description

SMG\$DELETE\_VIEWPORT deletes a viewport. The viewport is automatically “unpasted” from any pasteboards to which it is pasted. However, the virtual display associated with the deleted viewport has not been deleted. To view this virtual display, you must paste it to the pasteboard with the SMG\$PASTE\_VIRTUAL\_DISPLAY routine. To delete this virtual display, use the SMG\$DELETE\_VIRTUAL\_DISPLAY routine.

### Condition Values Returned

|                   |                                                  |
|-------------------|--------------------------------------------------|
| SS\$_NORMAL       | Normal successful completion.                    |
| SMG\$_INVDIS_ID   | Invalid <b>display-id</b> .                      |
| SMG\$_NO_WINASSOC | No viewport associated with the virtual display. |
| SMG\$_WRONUMARG   | Wrong number of arguments.                       |

## SMG\$DELETE\_VIRTUAL\_DISPLAY

---

### SMG\$DELETE\_VIRTUAL\_DISPLAY—Delete Virtual Display

The Delete Virtual Display routine deletes a virtual display.

#### Format

SMG\$DELETE\_VIRTUAL\_DISPLAY display-id

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

|                   |                     |
|-------------------|---------------------|
| <b>display-id</b> |                     |
| OpenVMS usage     | identifier          |
| type              | longword (unsigned) |
| access            | read only           |
| mechanism         | by reference        |

Specifies the virtual display to be deleted. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### Description

SMG\$DELETE\_VIRTUAL\_DISPLAY deletes a virtual display and removes it from any pasteboard on which it is pasted. It also deallocates any buffer space associated with the virtual display.

#### Condition Values Returned

|                 |                                                                          |
|-----------------|--------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                            |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                              |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal.                                      |
| SMG\$_WRONUMARG | Wrong number of arguments.                                               |
| SMG\$_NOTPASTED | The specified virtual display is not pasted to the specified pasteboard. |

Any condition values returned by LIB\$FREE\_VM.

---

## SMG\$DELETE\_VIRTUAL\_KEYBOARD—Delete Virtual Keyboard

The Delete Virtual Keyboard routine deletes a virtual keyboard.

### Format

SMG\$DELETE\_VIRTUAL\_KEYBOARD keyboard-id

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

|                    |                     |
|--------------------|---------------------|
| <b>keyboard-id</b> |                     |
| OpenVMS usage      | identifier          |
| type               | longword (unsigned) |
| access             | read only           |
| mechanism          | by reference        |

Specifies the virtual keyboard to be deleted. The **keyboard-id** argument is the address of an unsigned longword that contains the keyboard identifier.

**Keyboard-id** is returned by SMG\$CREATE\_VIRTUAL\_KEYBOARD.

### Description

SMG\$DELETE\_VIRTUAL\_KEYBOARD deletes a virtual keyboard. Any terminal attributes specified when the keyboard was created are reset to their previous values and the keypad mode (numeric or application) is reset to its original state. In addition, the channel is deassigned.

Because SMG\$ provides its own exit handlers, this routine should not be called from your own exit handler.

### Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVKBD_ID | Invalid <b>keyboard-id</b> .  |
| SMG\$_WRONUMARG | Wrong number of arguments.    |

## SMG\$DISABLE\_BROADCAST\_TRAPPING

---

### SMG\$DISABLE\_BROADCAST\_TRAPPING—Disable Broadcast Trapping

The Disable Broadcast Trapping routine disables trapping of broadcast messages for the specified terminal.

#### Format

SMG\$DISABLE\_BROADCAST\_TRAPPING pasteboard-id

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

|                      |                     |
|----------------------|---------------------|
| <b>pasteboard-id</b> |                     |
| OpenVMS usage        | identifier          |
| type                 | longword (unsigned) |
| access               | read only           |
| mechanism            | by reference        |

Specifies the pasteboard for the terminal to be affected. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

#### Description

SMG\$DISABLE\_BROADCAST\_TRAPPING disables trapping of broadcast messages for the specified terminal. SMG\$DISABLE\_BROADCAST\_TRAPPING deassigns the mailbox set with SMG\$SET\_BROADCAST\_TRAPPING, resets the terminal characteristics, and therefore allows the user to call LIB\$SPAWN. This routine must be used to disable any broadcast trapping set with the routine SMG\$SET\_BROADCAST\_TRAPPING.

When you disable broadcast trapping, any broadcast messages that have been queued to the terminal are lost. If you enable broadcast trapping with SMG\$SET\_BROADCAST\_TRAPPING but do not disable it with SMG\$DISABLE\_BROADCAST\_TRAPPING before the image exits, any messages that have been broadcast to the terminal are lost when the image exits.

Note that if both broadcast trapping and the trapping of unsolicited input are enabled, then both SMG\$DISABLE\_BROADCAST\_TRAPPING and SMG\$DISABLE\_UN SOLICITED\_INPUT must be invoked to deassign the mailbox.

#### Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_WRONUMARG | Wrong number of arguments.    |

Any condition value returned by \$QIOW.



## Example

```

10 !+
 !This VAX BASIC program creates three virtual displays on
 !one pasteboard.
 !
 !The first virtual display contains instructions for the user,
 !the second shows trapped unsolicited input, and the third
 !lists trapped broadcast messages. The program sits in an
 !infinite loop until the user types a CTRL/Z.
 !
 !When the program traps unsolicited input, both broadcast message
 !and unsolicited input trapping are disabled, and a subprocess
 !is spawned which executes the trapped user input.
 !
 !When control returns to the main process, broadcast trapping and
 !the trapping of unsolicited input are both reenabled. If the
 !unsolicited input which is trapped is a CTRL/Z, the program exits.
 !-

 OPTION TYPE = EXPLICIT

 !+
 !Declaration of all routines called by the main program.
 !-

 %INCLUDE "LIB$ROUTINES" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"
 %INCLUDE "SMG$ROUTINES" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"

 !+
 !Declaration of the two AST routines:
 !GET_MSG is called when a broadcast message is trapped
 !GET_INPUT is called when there is unsolicited input
 !GET_INPUT is the routine which spawns the subprocess
 !-

 EXTERNAL INTEGER GET_MSG
 EXTERNAL INTEGER GET_INPUT

 DECLARE LONG pb_id, ret_status, display_id, display2_id, display3_id, &
 key_id, key_tab_id, counter

 !+
 !Create a MAP area for variables which must be shared between the
 !main program and the AST routines.
 !-

 MAP (params) LONG disp_info(2), LONG keyboard_info(4), LONG done_flag

 DECLARE STRING CONSTANT top_label = "User Input"
 DECLARE STRING CONSTANT ins_label = "Instructions"
 DECLARE STRING CONSTANT msg_label = "Messages"
 DECLARE STRING CONSTANT instr_0 = "Type commands to fill INPUT display."
 DECLARE STRING CONSTANT instr_1 = "Type CTRL/T to fill MESSAGES display."
 DECLARE STRING CONSTANT instr_2 = "Type CTRL/Z to exit."
 DECLARE LONG CONSTANT advance = 1

 %INCLUDE "$SMGDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"
 %INCLUDE "$SMGMSG" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"

 !+
 !The done_flag variable is clear (0) unless the user input was
 !a CTRL/Z - in that case the program exits.
 !-
 done_flag = 0

 !+
 !Create the pasteboard and the virtual keyboard
 !-

```

## SMG\$DISABLE\_BROADCAST\_TRAPPING

```
ret_status = SMG$CREATE_PASTEBOARD (pb_id)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!This is one of the values which must be stored in the MAP area.
!-

disp_info(0) = pb_id

ret_status = SMG$CREATE_VIRTUAL_KEYBOARD (key_id)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$CREATE_KEY_TABLE (key_tab_id)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!Create the three virtual displays
!-

ret_status = SMG$CREATE_VIRTUAL_DISPLAY(3 BY REF, 75 BY REF, &
 display3_id, SMG$M_BORDER BY REF, SMG$M_REVERSE BY REF)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$CREATE_VIRTUAL_DISPLAY(6 BY REF, 75 BY REF, &
 display_id, SMG$M_BORDER BY REF, SMG$M_REVERSE BY REF)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$CREATE_VIRTUAL_DISPLAY(6 BY REF, 75 BY REF, &
 display2_id, SMG$M_BORDER BY REF, SMG$M_REVERSE BY REF)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!The disp_info and keyboard_info arrays are required in the MAP.
!-
disp_info(1) = display2_id

keyboard_info(0) = key_id
keyboard_info(1) = key_tab_id
keyboard_info(2) = display_id
keyboard_info(4) = pb_id

!+
!Put Label borders around the three virtual displays.
!-

ret_status = SMG$LABEL_BORDER (display3_id, ins_label,,, &
 SMG$M_BOLD BY REF, SMG$M_REVERSE BY REF)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$LABEL_BORDER (display_id, top_label,,, &
 SMG$M_BOLD BY REF,)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF
```

## SMG\$DISABLE\_BROADCAST\_TRAPPING

```
ret_status = SMG$LABEL_BORDER (display2_id, msg_label,,, &
 SMG$M_BOLD BY REF,)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!Fill the INSTRUCTIONS virtual display with user instructions.
!-

ret_status = SMG$PUT_LINE(display3_id, instr_0, &
 advance,,, smg$m_wrap_char)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$PUT_LINE(display3_id, instr_1, &
 advance,,, smg$m_wrap_char)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$PUT_LINE(display3_id, instr_2, &
 advance,,, smg$m_wrap_char)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!Paste the virtual displays to the screen.
!-

ret_status = SMG$PASTE_VIRTUAL_DISPLAY(display3_id, pb_id, &
 2 BY REF, 4 BY REF)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$PASTE_VIRTUAL_DISPLAY(display_id, pb_id, &
 8 BY REF, 4 BY REF)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$PASTE_VIRTUAL_DISPLAY(display2_id, pb_id, &
 18 BY REF, 4 BY REF)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!Enable the trapping of unsolicited input. GET_INPUT is the
!AST procedure that is called when unsolicited input is
!received. This AST has one parameter, passed as null.
!-

ret_status = SMG$ENABLE_UN SOLICITED_INPUT(pb_id, LOC(GET_INPUT))
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!Enable the trapping of broadcast messages. GET_MSG is the
!AST which is called when broadcast messages are received.
!This AST outputs the trapped message into the MESSAGES display.
!-
```

## SMG\$DISABLE\_BROADCAST\_TRAPPING

```
ret_status = SMG$SET_BROADCAST_TRAPPING(pb_id, LOC(GET_MSG))
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

!+
!This loop continually executes until done_flag is set to 1.
!Done_flag is set to 1 when the user input is a CTRL/Z.
!If done_flag is 1, delete the pasteboard and exit the program.
!-

Infinite_loop:
IF done_flag = 0 THEN
 GOTO infinite_loop
ELSE
 ret_status = SMG$DELETE_PASTEBOARD (pb_id)
 GOTO all_done
END IF

All_done:
END

20 !+
!Start of AST routine GET_INPUT. This AST is called whenever there
!is unsolicited input. The unsolicited input is displayed in the
!INPUT virtual display, and if this input is not CTRL/Z, a subprocess
!is spawned and the input command is executed. While this spawned
!subprocess is executing, broadcast and unsolicited input trapping
!are disabled.
!-

SUB GET_INPUT (paste_id, param, nl_1, nl_2, nl_3, nl_4)

MAP (params) LONG disp_info(2), LONG keyboard_info(4), LONG done_flag

DECLARE LONG z_status, status2, keybd_id, keybd_tab_id, disp_id, &
 pastebd, new_display, spawn_status
DECLARE WORD msg2_len
DECLARE STRING msg2
DECLARE LONG CONSTANT next_line = 1

%INCLUDE "SMG$ROUTINES" %FROM %LIBRARY "SYS$LIBRARY:ASIC$STARLET"
%INCLUDE "LIB$ROUTINES" %FROM %LIBRARY "SYS$LIBRARY:ASIC$STARLET"
%INCLUDE "$SMGMSG" %FROM %LIBRARY "SYS$LIBRARY:ASIC$STARLET"

EXTERNAL INTEGER GET_MSG
EXTERNAL INTEGER GET_INPUT

!+
!Assign to the local variables the values that were stored from
!the main program using the MAP area.
!-

keybd_id = keyboard_info(0)
keybd_tab_id = keyboard_info(1)
disp_id = keyboard_info(2)
pastebd = keyboard_info(3)

!+
!SMG$ENABLE_UNSOLICITED_INPUT does not read the input, it simply
!signals the specified AST when there is unsolicited input present.
!You must use SMG$READ_COMPOSED_LINE to actually read the input.
!
!At this time, we check to see if the unsolicited input was a CTRL/Z.
!If so, we skip over the program lines that spawn the subprocess and
!get ready to exit the program.
!-
```

## SMG\$DISABLE\_BROADCAST\_TRAPPING

```
status2 = SMG$READ_COMPOSED_LINE (keybd_id, keybd_tab_id, msg2,, &
 msg2_len, disp_id)
IF (status2 = SMG$_EOF) THEN
 GOTO Control_Z
END IF

IF (status2 AND 1%) = 0% THEN
 CALL LIB$STOP (status2 BY VALUE)
END IF

!+
!In order to spawn a subprocess, we must first disable
!unsolicited input trapping and broadcast trapping.
!-

status2 = SMG$DISABLE_UN SOLICITED_INPUT (pastebd)
IF (status2 AND 1%) = 0% THEN
 CALL LIB$STOP (status2 BY VALUE)
END IF

status2 = SMG$DISABLE_BROADCAST_TRAPPING (pastebd)
IF (status2 AND 1%) = 0% THEN
 CALL LIB$STOP (status2 BY VALUE)
END IF

!+
!Save the current screen so that it will not be destroyed when
!the subprocess is executing.
!-

status2 = SMG$SAVE_PHYSICAL_SCREEN (pastebd, new_display)
IF (status2 AND 1%) = 0% THEN
 CALL LIB$STOP (status2 BY VALUE)
END IF

!+
!Call LIB$SPAWN to create the subprocess, and pass the unsolicited
!input as the command line.
!-

spawn_status = LIB$SPAWN (msg2)

!+
!Restore the saved screen image.
!-

status2 = SMG$RESTORE_PHYSICAL_SCREEN (pastebd, new_display)
IF (status2 AND 1%) = 0% THEN
 CALL LIB$STOP (status2 BY VALUE)
END IF

!+
!Reenable broadcast trapping and unsolicited input trapping.
!-

status2 = SMG$ENABLE_UN SOLICITED_INPUT (pastebd, LOC(GET_INPUT))
IF (status2 AND 1%) = 0% THEN
 CALL LIB$STOP (status2 BY VALUE)
END IF

status2 = SMG$SET_BROADCAST_TRAPPING (pastebd, LOC(GET_MSG))
IF (status2 AND 1%) = 0% THEN
 CALL LIB$STOP (status2 BY VALUE)
END IF

!+
!Skip the steps which are performed if the unsolicited input
!was a CTRL/Z.
!-

GOTO Out_of_sub
```

## SMG\$DISABLE\_BROADCAST\_TRAPPING

```
Control_Z:
 !+
 !We should disable unsolicited input and broadcast trapping
 !before we leave the program.
 !-

 status2 = SMG$DISABLE_UNSOLICITED_INPUT (pastebd)
 IF (status2 AND 1%) = 0% THEN
 CALL LIB$STOP (status2 BY VALUE)
 END IF
 status2 = SMG$DISABLE_BROADCAST_TRAPPING (pastebd)
 IF (status2 AND 1%) = 0% THEN
 CALL LIB$STOP (status2 BY VALUE)
 END IF

 !+
 !Set the done_flag to 1 so that the main program knows we have
 !to exit.
 !-

 done_flag = 1
Out_of_sub:
 END SUB
30 !+
 !Start of AST routine GET_MSG. This AST is called whenever there
 !is a broadcast message. This routine prints the message in the
 !MESSAGES virtual display.
 !-

 SUB GET_MSG (paste_id, nl_1, nl_2, nl_3, nl_4)
 DECLARE LONG status1, pasteboard, second_disp
 DECLARE WORD msg_len
 DECLARE STRING msg
 DECLARE LONG CONSTANT forward = 1

 MAP (params) LONG disp_info(2), LONG keyboard_info(4)

 %INCLUDE "SMG$ROUTINES" %FROM %LIBRARY "SYS$LIBRARY:ASIC$STARLET"
 %INCLUDE "LIB$ROUTINES" %FROM %LIBRARY "SYS$LIBRARY:ASIC$STARLET"
 %INCLUDE "$SMGDEF" %FROM %LIBRARY "SYS$LIBRARY:ASIC$STARLET"
 %INCLUDE "$SMGMSG" %FROM %LIBRARY "SYS$LIBRARY:ASIC$STARLET"

 !+
 !Assign values to the local variables according to the values stored
 !in the MAP area.
 !-

 pasteboard = disp_info(0)
 second_disp = disp_info(1)

 !+
 !Print the trapped message in the MESSAGES display. If there are no
 !more messages, go back to the infinite loop in the main program.
 !-

 WHILE 1
 status1 = SMG$GET_BROADCAST_MESSAGE (pasteboard, msg, msg_len)
 IF (status1 = SMG$_NO_MORMSG) THEN
 GOTO Exitloop
 END IF
 IF (status1 AND 1%) = 0% THEN
 CALL LIB$STOP (status1 BY VALUE)
 END IF
```

## SMG\$DISABLE\_BROADCAST\_TRAPPING

```
 status1 = SMG$PUT_LINE (second_disp, msg, &
 forward,,, SMG$M_WRAP_CHAR)
 IF (status1 AND 1%) = 0% THEN
 CALL LIB$STOP (status1 BY VALUE)
 END IF
NEXT
Exitloop:
 END SUB
```

To run the example program, use the following commands.

```
$ BASIC TRAPPING
$ LINK TRAPPING
$ RUN TRAPPING
```

The output for this program is illustrated in the following figures. In Figure SMG-14, the program is waiting for either unsolicited input or broadcast messages.

**Figure SMG-14 Output Generated Before Any Input or Messages Are Trapped**

|                                                                                                       |
|-------------------------------------------------------------------------------------------------------|
| Instructions                                                                                          |
| Type commands to fill INPUT display.<br>Type CTRL/T to fill MESSAGES display.<br>Type CTRL/Z to exit. |
| User Input                                                                                            |
|                                                                                                       |
| Messages                                                                                              |
|                                                                                                       |

ZK-4805-GE

## SMG\$DISABLE\_BROADCAST\_TRAPPING

The output generated after the user types a Ctrl/T is shown in Figure SMG-15.

**Figure SMG-15 Output Generated After a Broadcast Message Is Trapped**

|                                                                                                       |
|-------------------------------------------------------------------------------------------------------|
| Instructions                                                                                          |
| Type commands to fill INPUT display.<br>Type CTRL/T to fill MESSAGES display.<br>Type CTRL/Z to exit. |
| User Input                                                                                            |
|                                                                                                       |
| Messages                                                                                              |
| FUTBAL::COLLINS_1 08:26:43 SAVE CPU=00:00:02.87 PF=401 IO=287 MEM=313                                 |

ZK-4806-GE

If the user types a command, that command is displayed in the INPUT display, and a subprocess is spawned. The output generated after the user types the command MAIL is shown in Figure SMG-16.



**Figure SMG–16 Output Generated After a Call to LIB\$SPAWN**

```
You have 3 new messages.
MAIL>
```

ZK-4807-GE

Once the subprocess completes execution, control is returned to the main process. At this point, the screen is repainted and the program continues to wait for broadcast messages or unsolicited input. The user must type a Ctrl/Z to exit the program.

## SMG\$DISABLE\_UNSOLICITED\_INPUT

---

### SMG\$DISABLE\_UNSOLICITED\_INPUT—Disable Unsolicited Input

The Disable Unsolicited Input routine disables the trapping of unsolicited input.

#### Format

SMG\$DISABLE\_UNSOLICITED\_INPUT pasteboard-id

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

|                      |                     |
|----------------------|---------------------|
| <b>pasteboard-id</b> |                     |
| OpenVMS usage        | identifier          |
| type                 | longword (unsigned) |
| access               | read only           |
| mechanism            | by reference        |

Specifies the keyboard (associated with the specified pasteboard) for which unsolicited input is being disabled. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

#### Description

SMG\$DISABLE\_UNSOLICITED\_INPUT disables unsolicited input ASTs for the specified pasteboard. SMG\$DISABLE\_UNSOLICITED\_INPUT deassigns the mailbox set with SMG\$ENABLE\_UNSOLICITED\_INPUT, resets the terminal characteristics, and therefore allows the user to call LIB\$SPAWN. This routine must be used to disable any unsolicited input trapping enabled with the SMG\$ENABLE\_UNSOLICITED\_INPUT routine.

Note that if both unsolicited input trapping and the trapping of broadcast messages are enabled, then both SMG\$DISABLE\_UNSOLICITED\_INPUT and SMG\$DISABLE\_BROADCAST\_TRAPPING must be invoked in order to deassign the mailbox.

#### Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_WRONUMARG | Wrong number of arguments.     |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |

Any condition values returned by \$QIOW.

### Example

For an example of using SMG\$DISABLE\_UNSOLICITED\_INPUT, see the example for the routine SMG\$DISABLE\_BROADCAST\_TRAPPING.

---

## SMG\$DRAW\_CHAR—Draw a Character in a Virtual Display

The Draw a Character in a Virtual Display routine draws a character at the specified position in a virtual display.

### Format

SMG\$DRAW\_CHAR display-id ,flags [,row] [,column] [,rendition-set] [,rendition-complement]

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the virtual display. The **display-id** argument is the address of an unsigned longword containing this identifier.

#### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask indicating the character to be drawn. The **flags** argument is the address of an unsigned longword that contains the flag. **Flags** accepts the following character values:

- SMG\$M\_UP
- SMG\$M\_DOWN
- SMG\$M\_LEFT
- SMG\$M\_RIGHT

Note that you may perform a logical OR operation to draw T characters, corner characters, cross characters, and so forth. A value of 0 draws a diamond character.

#### row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Optional row number specifying the row position at which the specified character is drawn. The **row** argument is the address of a signed longword containing the

row number. If **row** is omitted, the character is drawn at the row position of the current virtual cursor.

#### **column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Optional column number specifying the column position at which the specified character is drawn. The **column** argument is the address of a signed longword containing the column number. If **column** is omitted, the character is drawn at the column position of the current virtual cursor.

In case that the **column** is on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also if it's on the left portion of a Chinese character, the right portion of the character will become an undefined character.

#### **rendition-set**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                          |                                                                                                                           |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK                             | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD                              | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE                           | Displays characters in reverse video, that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE                         | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE                         | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$M_USER1<br>through SMG\$M_<br>USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

#### **rendition-complement**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented

## SMG\$DRAW\_CHAR

with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

### Description

SMG\$DRAW\_CHAR draws a designated character at the specified position in the specified virtual display. Note that this routine does not change the position of the virtual cursor. The characters drawn depend on the type of terminal. For example, SMG\$ uses the terminal's line-drawing character set if possible. If that is not available, SMG\$ will use the characters +, -, and | to draw a line.

In case that the Chinese language character set is used, and the **column** is on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also if it's on the left portion of a Chinese character, the right portion of the character will become an undefined character.

### Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVCOL    | Invalid column number.        |
| SMG\$_INVROW    | Invalid row number.           |
| SMG\$_WRONUMARG | Wrong number of arguments.    |

### Example

```
C+
C This DEC Fortran example demonstrates the use of
C SMG$DRAW_CHAR to use the terminal line drawing
C characters.
C-
 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'

 s = SMG$CREATE_PASTEBOARD(p_id)
 IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
 s = SMG$CREATE_VIRTUAL_DISPLAY(17,7,d_id,SMG$_M_BORDER)
 IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
 s = SMG$PASTE_VIRTUAL_DISPLAY(d_id,p_id,4,30)
 IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
 s = SMG$SET_CURSOR_REL(d_id,1,3)
 IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
```

## SMG\$DRAW\_CHAR

```
s = SMG$DRAW_CHAR(d_id,SMG$M_UP,1,4,SMG$M_BOLD)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_DOWN,2,4,0,SMG$M_REVERSE)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_LEFT,3,4,SMG$M_BLINK)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_RIGHT,4,4,0,0)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_UP + SMG$M_DOWN,5)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_UP + SMG$M_LEFT,6)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_UP + SMG$M_RIGHT,7)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_DOWN + SMG$M_LEFT,8)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_DOWN + SMG$M_RIGHT,9)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_LEFT + SMG$M_RIGHT,10)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_UP + SMG$M_DOWN + SMG$M_LEFT,11)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_UP + SMG$M_DOWN + SMG$M_RIGHT,12)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_DOWN + SMG$M_LEFT + SMG$M_RIGHT,13)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_UP + SMG$M_LEFT + SMG$M_RIGHT,14)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,SMG$M_UP + SMG$M_DOWN + SMG$M_RIGHT +
1 SMG$M_LEFT, 15)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))
s = SMG$DRAW_CHAR(d_id,0,16)
IF (.NOT. s) CALL LIB$SIGNAL(%VAL(s))

END
```

This example generates line drawing characters in a single column.

---

## SMG\$DRAW\_LINE—Draw a Line

The Draw a Line routine draws a horizontal or vertical line.

### Format

```
SMG$DRAW_LINE display-id ,start-row ,start-column ,end-row ,end-column [,rendition-set]
 [,rendition-complement]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### **display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display on which the line is to be drawn. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### **start-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row at which to begin drawing the line. The **start-row** argument is the address of a signed longword that contains the row number at which to begin drawing the line.



**start-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column at which to begin drawing the line. The **start-column** argument is the address of a signed longword that contains the column number at which to begin drawing the line.

In the case that the line is horizontal and the **start-column** begins on the right portion of a Chinese character, the left portion of the character will become an undefined character. If the line is vertical and the **start-column** is on the left or right portion of a Chinese characters for each row, the remaining portions of the characters becomes undefined characters.

**end-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row at which the drawn line ends. The **end-row** argument is the address of a signed longword that contains the row number at which the drawn line ends.

**end-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column at which the drawn line ends. The **end-column** argument is the address of a signed longword that contains the column number at which the drawn line ends.

In case that the line is horizontal and the **end-column** ends on the left portion of a Chinese character, the right portion of the character will become an undefined character. If the line is vertical and the **end-column** is on the left or right portion of a Chinese characters for each row, the remaining portions of the characters will become undefined characters.

## SMG\$DRAW\_LINE

### rendition-set

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

SMG\$M\_BLINK Displays blinking characters.  
SMG\$M\_BOLD Displays characters in higher-than-normal intensity.  
SMG\$M\_REVERSE Displays characters in reverse video, that is, using the opposite of the default rendition of the virtual display.  
SMG\$M\_UNDERLINE Displays underlined characters.  
SMG\$M\_INVISIBLE Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard.  
SMG\$M\_USER1 through SMG\$M\_USER8 Displays user-defined attributes.

The **display-id** argument must be specified when you use the **rendition-set** argument.

### rendition-complement

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |

| Set | Complement | Action        |
|-----|------------|---------------|
| 1   | 1          | Attribute off |

## Description

SMG\$DRAW\_LINE draws a line from a specified starting row and column to a specified ending row and column. Note that this routine does not change the virtual cursor position. You can draw only horizontal or vertical lines. The characters used to draw the line depend on the type of terminal. If possible, SMG\$ uses the terminal's line-drawing character set. If that is not available, SMG\$ uses the characters +, -, and | to draw the line.

In case that the Chinese language character set is used, and the drawn line is on the left or right portion of the Chinese characters, the remaining portions of the characters will become undefined characters.

## Condition Values Returned

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                               |
| SMG\$_INVCOL    | Invalid column number. The specified column is outside the virtual display. |
| SMG\$_INVROW    | Invalid row number. The specified row is outside the virtual display.       |
| SMG\$_DIALINNOT | Diagonal line not allowed.                                                  |
| SMG\$_WRONUMARG | Wrong number of arguments.                                                  |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                                 |

## Example

```

C+
C This DEC Fortran example program demonstrates the use of SMG$DRAW_LINE.
C-

 INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
 INTEGER SMG$PASTE_VIRTUAL_DISPLAY, SMG$DRAW_LINE
 INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS, BORDER, STATUS

C+
C First, create the virtual display using SMG$CREATE_VIRTUAL_DISPLAY.
C To give it a border, set BORDER = 1. No border would be BORDER = 0.
C-

 ROWS = 7
 COLUMNS = 50
 BORDER = 1

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

```

## SMG\$DRAW\_LINE

```
C+
C Draw a vertical line using SMG$DRAW_LINE.
C Start at row 2, column 20. End at row 6.
C-
 STATUS = SMG$DRAW_LINE (DISPLAY1, 2, 20, 6, 20)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Now, use SMG$DRAW_LINE to draw a vertical line.
C Start at row 6, column 40. End at row 2.
C This is similar to the line drawn above, but we are drawing the
C line in the reverse direction.
C-
 STATUS = SMG$DRAW_LINE (DISPLAY1, 6, 40, 2, 40)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

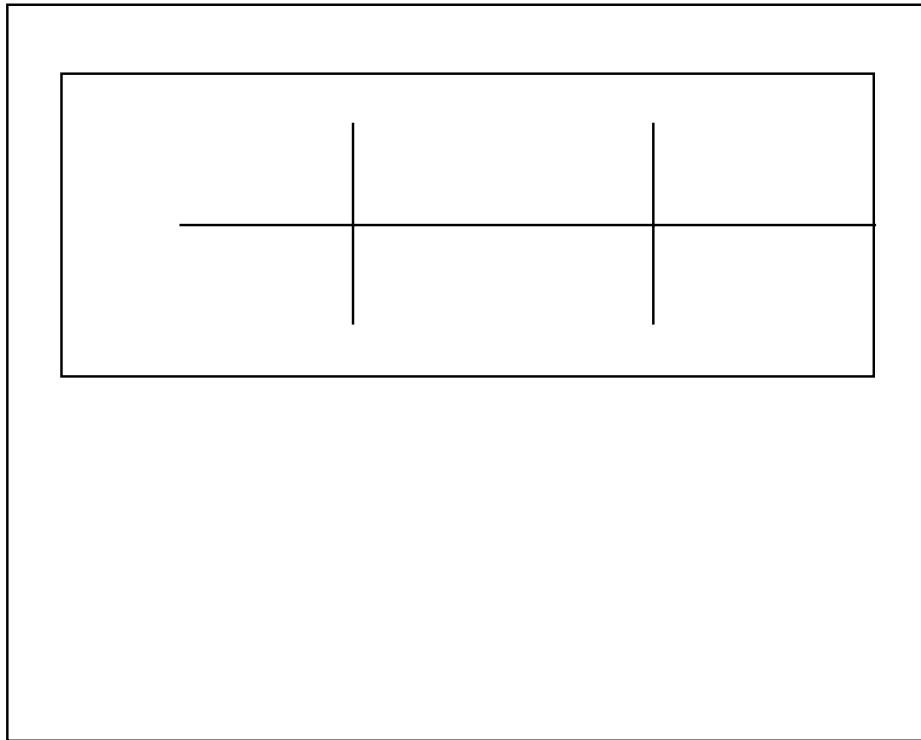
C+
C Draw a horizontal line now, again calling SMG$DRAW_LINE.
C Start at row 4, column 8. End at column 50.
C-
 STATUS = SMG$DRAW_LINE (DISPLAY1, 4, 8, 4, 50)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Paste the virtual display using SMG$PASTE_VIRTUAL_DISPLAY.
C-
 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 END
```

The output generated by this FORTRAN example is shown in Figure SMG-17.

Figure SMG-17 Output Generated by FORTRAN Program Calling SMG\$DRAW\_LINE



ZK-4110-GE

## SMG\$DRAW\_RECTANGLE

---

### SMG\$DRAW\_RECTANGLE—Draw a Rectangle

The Draw a Rectangle routine draws a rectangle.

#### Format

```
SMG$DRAW_RECTANGLE display-id ,start-row ,start-column ,end-row ,end-column [,rendition-set]
 [,rendition-complement]
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display on which the rectangle is to be drawn. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### **start-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row number of the top left-hand corner of the rectangle. The **start-row** argument is the address of a signed longword that contains the row number of the top left-hand corner of the rectangle.

**start-column**

OpenVMS usage longword\_signed  
 type longword (signed)  
 access read only  
 mechanism by reference

Specifies the column number of the top left-hand corner of the rectangle. The **start-column** argument is the address of a signed longword that contains the column number of the top left-hand corner of the rectangle.

In case that the **start-column** is on the left or right portion of a Chinese characters for each row, the remaining portions of the characters will become undefined characters.

**end-row**

OpenVMS usage longword\_signed  
 type longword (signed)  
 access read only  
 mechanism by reference

Specifies the row number of the bottom right-hand corner of the rectangle. The **end-row** argument is the address of a signed longword that contains the row number of the bottom right-hand corner of the rectangle.

**end-column**

OpenVMS usage longword\_signed  
 type longword (signed)  
 access read only  
 mechanism by reference

Specifies the column number of the bottom right-hand corner of the rectangle. The **end-column** argument is the address of a signed longword that contains the column number of the bottom right-hand corner of the rectangle.

In case that the **end-column** is on the left or right portion of a Chinese characters for each row, the remaining portions of the characters will become undefined characters.

**rendition-set**

OpenVMS usage mask\_longword  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                 |                                                                                                                    |
|-----------------|--------------------------------------------------------------------------------------------------------------------|
| SMG\$_BLINK     | Displays blinking characters.                                                                                      |
| SMG\$_BOLD      | Displays characters in higher-than-normal intensity.                                                               |
| SMG\$_REVERSE   | Displays characters in reverse video, that is, using the opposite of the default rendition of the virtual display. |
| SMG\$_UNDERLINE | Displays underlined characters.                                                                                    |

## SMG\$DRAW\_RECTANGLE

SMG\$M\_INVISIBLE Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard.

SMG\$M\_USER1 through SMG\$M\_USER8 Displays user-defined attributes.

The **display-id** argument must be specified when you use the **rendition-set** argument.

### rendition-complement

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

## Description

SMG\$DRAW\_RECTANGLE draws a rectangle in a virtual display, given the position of the upper left-hand corner and the lower right-hand corner. Note that this routine does not change the virtual cursor position. The characters used to draw the lines making up the rectangle depend on the type of terminal. If possible, SMG\$ uses the terminal's line-drawing character set. If that is not available, SMG\$ uses the characters +, -, and | to draw the lines.

In case that the Chinese language character set is used, and the drawn rectangle is on the left or right portion of a Chinese characters for each row, the remaining portions of the characters will become undefined characters.



## Condition Values Returned

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                               |
| SMG\$_INVCOL    | Invalid column number. The specified column is outside the virtual display. |
| SMG\$_INVROW    | Invalid row number. The specified row is outside the virtual display.       |
| SMG\$_WRONUMARG | Wrong number of arguments.                                                  |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                                 |

## Example

```

C+
C This DEC Fortran example program demonstrates the use of
C SMG$DRAW_RECTANGLE.
C
C This routine creates a virtual display and uses SMG$DRAW_RECTANGLE
C to draw a rectangle inside the bordered virtual display.
C-

C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-

 INCLUDE '($SMGDEF)'
 INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
 INTEGER SMG$PASTE_VIRTUAL_DISPLAY, SMG$DRAW_RECTANGLE
 INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS, STATUS

C+
C Create a virtual display with a border by calling
C SMG$CREATE_VIRTUAL_DISPLAY.
C-

 ROWS = 7
 COLUMNS = 50

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$CREATE_PASTEBOARD to create the pasteboard.
C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Using SMG$DRAW_RECTANGLE, draw a rectangle inside the bordered region.
C-

 STATUS = SMG$DRAW_RECTANGLE (DISPLAY1, 2, 10, 6, 20)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Paste the virtual display by calling SMG$PASTE_VIRTUAL_DISPLAY.
C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

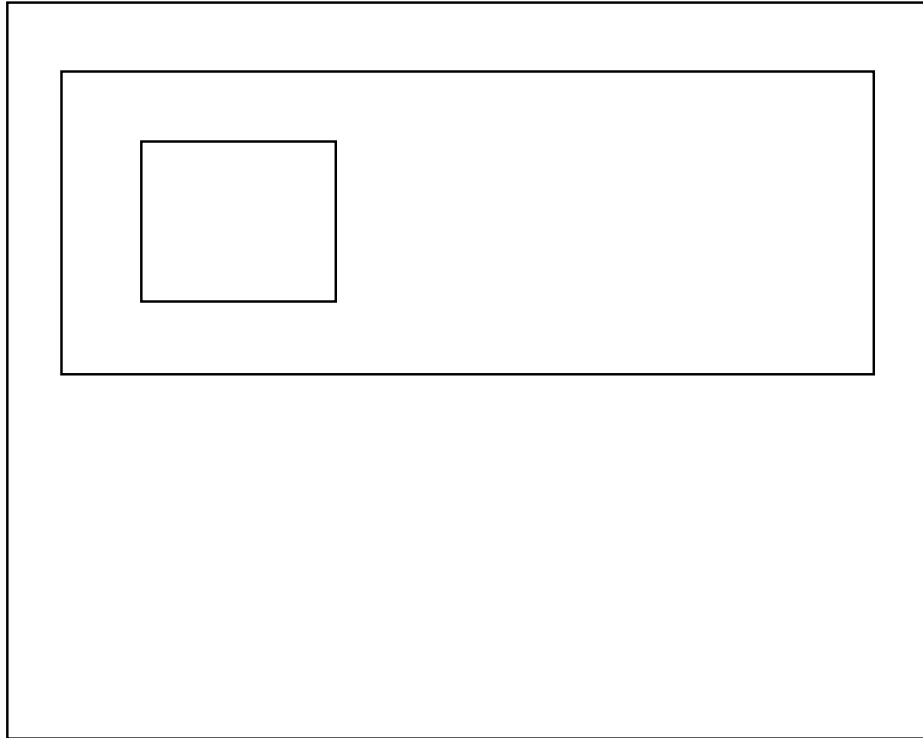
 END

```

The output generated by this FORTRAN example is shown in Figure SMG-18.

# SMG\$DRAW\_RECTANGLE

Figure SMG-18 Output Generated by FORTRAN Program Calling SMG\$DRAW\_RECTANGLE



ZK-4111-GE

---

## SMG\$ENABLE\_UNSOLICITED\_INPUT—Enable Unsolicited Input

The Enable Unsolicited Input routine detects unsolicited input and calls an AST routine in response.

### Format

```
SMG$ENABLE_UNSOLICITED_INPUT pasteboard-id ,AST-routine [,AST-argument]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### **pasteboard-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard for which unsolicited input is being enabled. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_VIRTUAL\_PASTEBOARD.

#### **AST-routine**

|               |                 |
|---------------|-----------------|
| OpenVMS usage | ast_procedure   |
| type          | procedure value |
| access        | read only       |
| mechanism     | by value        |

AST routine to be called upon receipt of unsolicited input at the terminal. The **AST-routine** argument contains the routine's procedure value. SMG\$ENABLE\_UNSOLICITED\_INPUT detects the presence of unsolicited input and calls the AST routine with six arguments: the **pasteboard-id**, the **AST-argument**, R0, R1, PC, and PSL (on VAX systems) or PS (on AXP systems). The AST routine arguments are shown in Figure SMG-19.

# SMG\$ENABLE\_UNSOLICITED\_INPUT

Figure SMG–19 AST Routine Arguments

|               |
|---------------|
| Pasteboard ID |
| AST Argument  |
| R0            |
| R1            |
| PC            |
| PSL           |

ZK-4802-GE

## AST-argument

OpenVMS usage user\_arg  
type longword (unsigned)  
access read only  
mechanism by value

A value to be passed to the AST routine. The **AST-argument** argument contains the value to be passed to the AST routine.

## Description

SMG\$ENABLE\_UNSOLICITED\_INPUT detects the presence of unsolicited input and calls an AST routine in response.

Note that this routine does not read any input characters; it merely calls an AST routine to “notify” the application that it should issue a read operation with SMG\$READ\_COMPOSED\_LINE, SMG\$READ\_KEYSTROKE or SMG\$READ\_STRING. It is up to you to read the unsolicited input.

SMG\$ENABLE\_UNSOLICITED\_INPUT establishes a mailbox that receives messages when terminal-related events occur that require the attention of the user image. This mailbox carries status messages, not terminal data, from the driver to the user program. This status message is sent to the mailbox when there is unsolicited data in the type-ahead buffer. In this case, the user process enters into a dialogue with the terminal after an unsolicited data message arrives. Once this dialogue is complete, the Screen Management Facility reenables the unsolicited data message function on the last I/O exchange. Only one message is sent between read operations. (The sysgen parameters DEFMBXBUFQUO and DEFMBXMXMSG are used when creating the mailbox.)

## Condition Values Returned

SS\$\_NORMAL Normal successful completion.  
SMG\$\_WRONUMARG Wrong number of arguments.  
SMG\$\_INVPAS\_ID Invalid **pasteboard-id**.

Any condition values returned by \$QIOW.

### Example

For an example using SMG\$ENABLE\_UNSOLICITED\_INPUT, see the example for the routine SMG\$DISABLE\_BROADCAST\_TRAPPING.

## SMG\$END\_DISPLAY\_UPDATE

---

### SMG\$END\_DISPLAY\_UPDATE—End Display Update

The End Display Update routine ends update batching for a virtual display.

#### Format

SMG\$END\_DISPLAY\_UPDATE display-id

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

|                   |                     |
|-------------------|---------------------|
| <b>display-id</b> |                     |
| OpenVMS usage     | identifier          |
| type              | longword (unsigned) |
| access            | read only           |
| mechanism         | by reference        |

Specifies the virtual display to be affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### Description

SMG\$END\_DISPLAY\_UPDATE and SMG\$BEGIN\_DISPLAY\_UPDATE work together to control the batching of output operations on a given virtual display. Each call to SMG\$BEGIN\_DISPLAY\_UPDATE increments a “batch count,” while each call to SMG\$END\_DISPLAY\_UPDATE decrements this count. When the batch count reaches 0, the virtual display is updated with all operations done under batching, and written to the pasteboard if the virtual display is pasted.

Calling SMG\$END\_DISPLAY\_UPDATE when the batch count is zero is a valid operation; therefore a success status is returned.

#### Condition Values Returned

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                   |
| SMG\$_BATWASOFF | Successful completion. Note that batching was already off.      |
| SMG\$_BATSTIPRO | Successful completion. Note that batching is still in progress. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                     |
| SMG\$_WRONUMARG | Wrong number of arguments.                                      |

---

## SMG\$END\_PASTEBOARD\_UPDATE—End Pasteboard Update

The End Pasteboard Update routine ends update batching for a pasteboard.

### Format

SMG\$END\_PASTEBOARD\_UPDATE pasteboard-id

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

|                      |                     |
|----------------------|---------------------|
| <b>pasteboard-id</b> |                     |
| OpenVMS usage        | identifier          |
| type                 | longword (unsigned) |
| access               | read only           |
| mechanism            | by reference        |

Specifies the pasteboard on which the batch count is to be decremented. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD. If the batch count reaches 0, all buffered output for the specified pasteboard is written out.

### Description

SMG\$END\_PASTEBOARD\_UPDATE and SMG\$BEGIN\_PASTEBOARD\_UPDATE work together to control the batching of output operations on a given pasteboard. Each call to SMG\$BEGIN\_PASTEBOARD\_UPDATE increments a “batch count,” while each call to SMG\$END\_PASTEBOARD\_UPDATE decrements this count. When the batch count reaches 0, the pasteboard is written to the screen.

Calling SMG\$END\_PASTEBOARD\_UPDATE when the batch count is 0 is a valid operation; a success status is returned.

### Condition Values Returned

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                   |
| SMG\$_BATWASOFF | Successful completion. Note that batching was already off.      |
| SMG\$_BATSTIPRO | Successful completion. Note that batching is still in progress. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                     |
| SMG\$_WRONUMARG | Wrong number of arguments.                                      |

## SMG\$ERASE\_CHARS

---

### SMG\$ERASE\_CHARS—Erase Characters

The Erase Characters routine erases columns in a virtual display by replacing them with blanks.

#### Format

SMG\$ERASE\_CHARS display-id ,number-of-columns ,start-row ,start-column

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display from which columns will be erased. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### **number-of-columns**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the number of columns to be replaced with blanks. The **number-of-columns** argument is the address of a signed longword that contains the number of columns to be replaced with blanks.



**start-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row on which the erase operation begins. The **start-row** argument is the address of a signed longword that contains the number of the row at which the erasure is to begin.

**start-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column on which the erase operation begins. The **start-column** argument is the address of a signed longword that contains the number of the column at which the erasure is to begin.

In case that the start-column begins on the right portion of a Chinese character, the right portion of the character is overridden by space, and the left portion of the character will become an undefined character.

**Description**

SMG\$ERASE\_CHARS erases columns in a virtual display by replacing them with blanks. The remaining text in the display is not moved. An erase operation is limited to the specified line. If **number-of-columns** is greater than the number of columns remaining in the line, all columns from the specified starting position to the end of the line are erased. This routine leaves the virtual cursor at the position of the first columns erased.

In case Chinese language character set is used, and the start column begins on the right portion of a Chinese character, the right portion of the character is overridden by space and the left portion of the character will become an undefined character. Also, if the erasure terminates on the left portion of a Chinese character, the right portion of the character will become an undefined character.

## SMG\$ERASE\_CHARS

### Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVCOL    | Invalid column.               |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .   |
| SMG\$_INVROW    | Invalid row.                  |
| SMG\$_WRONUMARG | Wrong number of arguments.    |

### Example

```
C+
C This DEC Fortran example demonstrates the use of SMG$ERASE_CHARS.
C
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'

C+
C Create a virtual display with a border by calling
C SMG$CREATE_VIRTUAL_DISPLAY.
C-

 ROWS = 7
 COLUMNS = 50

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Using SMG$PUT_CHARS, put data in the virtual display.
C-

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This virtual display has 7 rows and 50 columns.', 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This is a bordered virtual display.', 4, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

## SMG\$ERASE\_CHARS

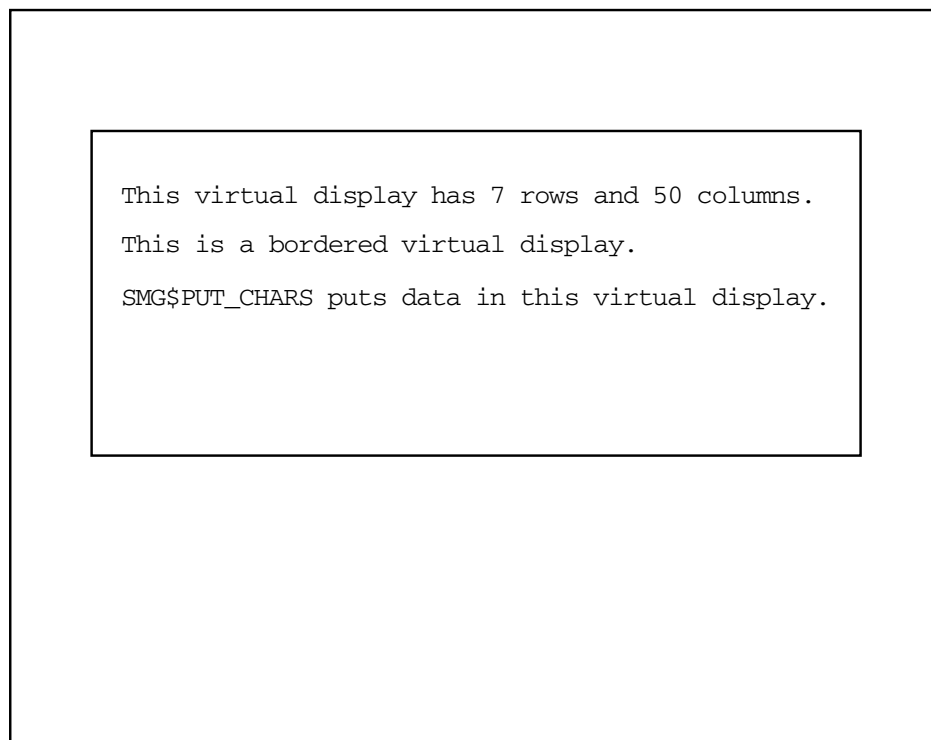
```
C+
C Erase 4 characters on row 4 starting from character (column) 14 by
C calling SMG$ERASE_CHARS. This will remove the characters "rder"
C from the word "bordered".
C-

 STATUS = SMG$ERASE_CHARS (DISPLAY1, 4, 4, 14)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 END
```

The initial output generated by this FORTRAN example program is shown in Figure SMG-20.

**Figure SMG-20 Output Before the Call to SMG\$ERASE\_CHARS**

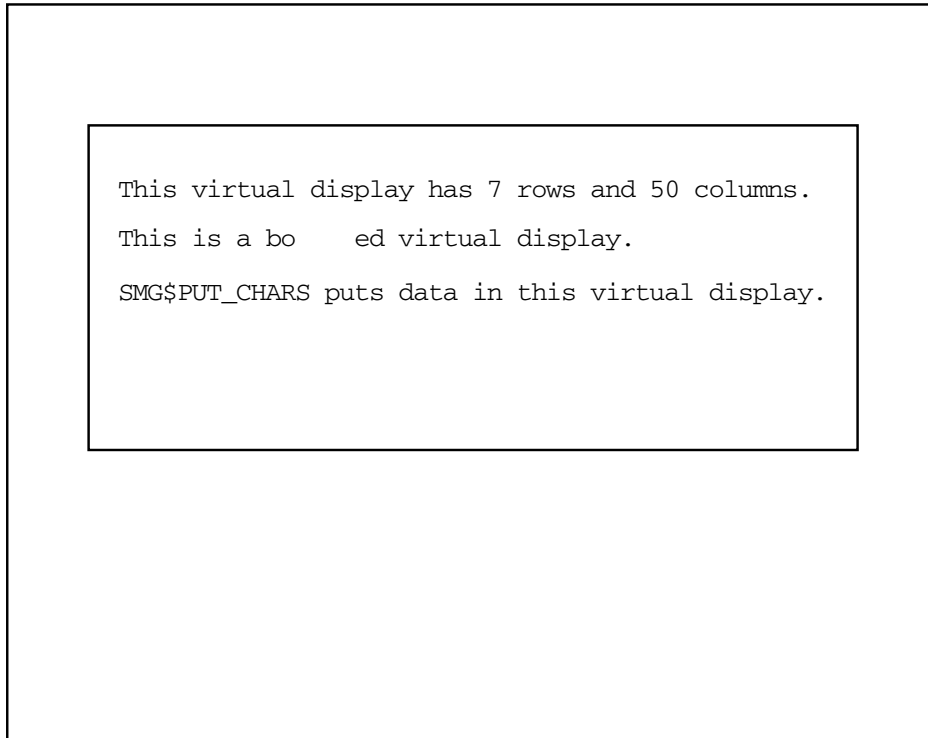


ZK-4105-GE

The output generated after the call to SMG\$ERASE\_CHARS is shown in Figure SMG-21.

## SMG\$ERASE\_CHARS

Figure SMG-21 Output After the Call to SMG\$ERASE\_CHARS



ZK-4113-GE

---

## SMG\$ERASE\_COLUMN—Erase Column from Display

The Erase Column From Display routine erases the specified portion of the virtual display from the given position to the end of the column.

### Format

```
SMG$ERASE_COLUMN display-id [,start-row] [,column-number] [,end-row]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the virtual display to be affected. The **display-id** argument is the address of an unsigned longword containing this virtual display identifier.

#### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Optional line number at which the erase operation begins. The **start-row** argument is the address of a signed longword that contains the specified line number. If this argument is omitted, the **column-number** argument is ignored and the erase operation begins at the current location of the virtual cursor for that virtual display.

#### column-number

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Optional column number at which the erase operation begins. The **column-number** argument is the address of a signed longword that contains the specified column number. If this argument is omitted, the **start-row** argument is ignored and the erase operation begins at the current location of the virtual cursor for that virtual display.

In the case that the **column-number** is on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if it's on the left portion of a Chinese character, the right portion of the character will become an undefined character.

## SMG\$ERASE\_COLUMN

### **end-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Optional row number at which the erase operation ends. The **end-row** argument is the address of a signed longword that contains the specified row number.

### **Description**

SMG\$ERASE\_COLUMN lets you erase a column of the virtual display from the specified position to the end of the column. If the position is not specified, the erase operation begins at the current position of the virtual cursor in the specified virtual display. After the erase operation has completed, this routine leaves the virtual cursor at the position of the first character erased.

In the case that the Chinese language character set is used, if the erasure operation occurs on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if it occurs on the left portion of a Chinese character, the right portion of the character will become an undefined character.

### **Condition Values Returned**

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                               |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                                 |
| SMG\$_INVCOL    | Invalid column number. The specified column is outside the virtual display. |
| SMG\$_INVROW    | Invalid row number. The specified row is outside the virtual display.       |
| SMG\$_WRONUMARG | Wrong number of arguments.                                                  |

---

## SMG\$ERASE\_DISPLAY—Erase Virtual Display

The Erase Virtual Display routine erases all or part of a virtual display by replacing text characters with blanks.

### Format

```
SMG$ERASE_DISPLAY display-id [,start-row] [,start-column] [,end-row] [,end-column]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display to be erased. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row at which the erase operation begins. The **start-row** argument is the address of a signed longword that contains the number of the row at which the erasure begins.

If the **start-row** argument is not specified, **start-column** is also ignored and the entire virtual display is erased. If you do not specify **start-row** and **start-column**, then **end-row** and **end-column** are ignored and the entire virtual display is erased.

#### start-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column at which the erase operation begins. The **start-column** argument is the address of a signed longword that contains the number of the column at which the erasure begins.

## SMG\$ERASE\_DISPLAY

If the **start-column** argument is not specified, **start-row** is also ignored and the entire virtual display is erased. If you do not specify **start-row** and **start-column**, then **end-row** and **end-column** are ignored and the entire virtual display is erased.

In case that the **start-column** begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

### end-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row at which the erase operation ends, that is, the last row to be erased. The **end-row** argument is the address of a signed longword that contains the number of the last row to be erased.

If the **end-row** argument is not specified, **end-column** is also ignored and all remaining rows in the display are erased.

### end-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column at which the erase operation ends, that is, the last column to be erased. The **end-column** argument is the address of a signed longword that contains the number of the last column to be erased.

If the **end-column** argument is not specified, **end-row** is also ignored and all remaining columns in the display are erased.

In case that the **end-column** terminates on the left portion of a Chinese character, the right portion of the character will become an undefined character.

## Description

SMG\$ERASE\_DISPLAY causes all or part of a virtual display to be erased by replacing text characters with blanks. If omitted, the starting positions default to 1,1. The ending positions default to the last row or column in the display. Thus, to erase the entire virtual display, you need only pass the **display-id**. This routine leaves the virtual cursor at the start of the erased position. If the entire display is erased, the virtual cursor is left at position 1,1.

In case Chinese language character set is used and the **start-column** begins on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if the erasure terminates on the left portion of a Chinese character, the right portion of the character will become an undefined character.



## Condition Values Returned

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                               |
| SMG\$_INVCOL    | Invalid column number. The specified column is outside the virtual display. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                                 |
| SMG\$_INVROW    | Invalid row number. The specified row is outside the virtual display.       |
| SMG\$_WRONUMARG | Wrong number of arguments.                                                  |

## Example

```

C+
C This DEC Fortran example program demonstrates the use of SMG$ERASE_DISPLAY.
C-

 IMPLICIT INTEGER (A-Z)

C+
C Call SMG$CREATE_VIRTUAL_DISPLAY to create the virtual
C display. To give it a border, set BORDER = 1.
C No border would be BORDER = 0.
C-

 ROWS = 7
 COLUMNS = 50
 BORDER = 1

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Using SMG$CREATE_PASTEBOARD, create the pasteboard.
C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$PUT_CHARS to put data in the virtual display.
C-

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This virtual display has 7 rows and 50 columns.', 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This is a bordered virtual display.', 4, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Paste the virtual display by calling SMG$PASTE_VIRTUAL_DISPLAY.
C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$ERASE_DISPLAY to erase the display from row 2,
C column 6, through row 4, column 28.
C-

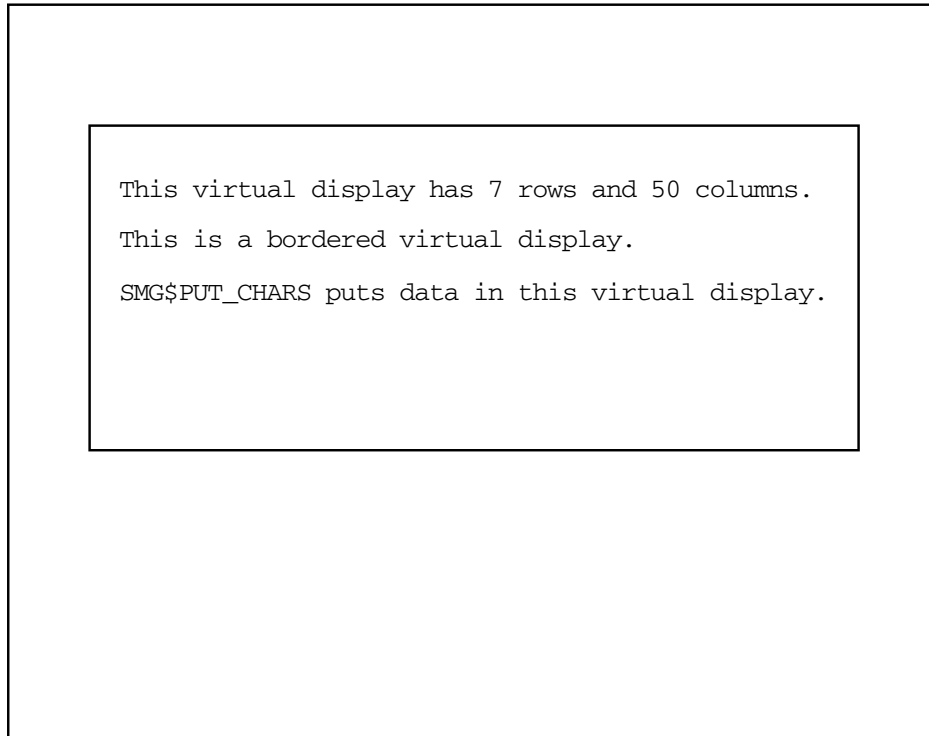
```

## SMG\$ERASE\_DISPLAY

```
STATUS = SMG$ERASE_DISPLAY (DISPLAY1, 2, 6, 4, 28)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
END
```

The initial display output by this FORTRAN program is shown in Figure SMG-22.

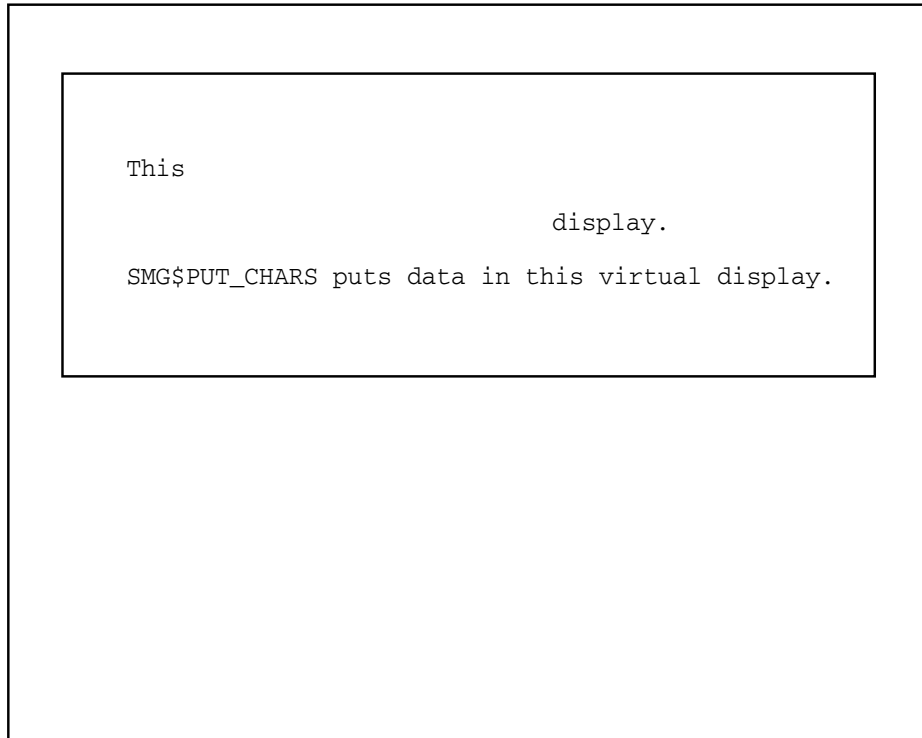
**Figure SMG-22 Initial Output of FORTRAN Program Calling SMG\$ERASE\_DISPLAY**



ZK-4105-GE

This output displayed after the call to SMG\$ERASE\_DISPLAY is shown in Figure SMG-23.

Figure SMG-23 Output Displayed After the Call to SMG\$ERASE\_DISPLAY



ZK-4115-GE

## SMG\$ERASE\_LINE

---

### SMG\$ERASE\_LINE—Erase Line

The Erase Line routine erases all or part of a line in a virtual display.

#### Format

SMG\$ERASE\_LINE display-id [,start-row] [,start-column]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display to be affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the line at which the erase operation starts. The **start-row** argument is the address of a signed longword that contains the number of the row at which the erasure starts. If omitted, **start-column** is also ignored and the current cursor position is used.

##### start-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column at which the erase operation starts. The **start-column** argument is the address of a signed longword that contains the number of the column at which the erasure starts. If omitted, **start-row** is also ignored and the current cursor position is used.

## Description

SMG\$ERASE\_LINE erases a line from the specified starting position to the end of the line. If you do not specify a starting position, SMG\$ERASE\_LINE erases text from the current virtual cursor position to the end of the line. This routine leaves the virtual cursor at the start of the erased portion.

In case Chinese language character set is used and the **start-column** begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

## Condition Values Returned

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                               |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                                 |
| SMG\$_INVCOL    | Invalid column number. The specified column is outside the virtual display. |
| SMG\$_INVROW    | Invalid row number. The specified row is outside the virtual display.       |
| SMG\$_WRONUMARG | Wrong number of arguments.                                                  |

## Example

```

C+
C This DEC Fortran example program demonstrates the use of
C SMG$ERASE_LINE.
C-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'

C+
C Use SMG$CREATE_VIRTUAL_DISPLAY to create a virtual display
C with a border.
C-

 ROWS = 7
 COLUMNS = 50

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$_M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Put data in the virtual display by calling SMG$PUT_CHARS.
C-

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This virtual display has 7 rows and 50 columns.', 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This is a bordered virtual display.', 4, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

```

## SMG\$ERASE\_LINE

```
 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$ERASE_LINE to erase line 2, and then again to
C erase the last 4 words on line 4.
C-

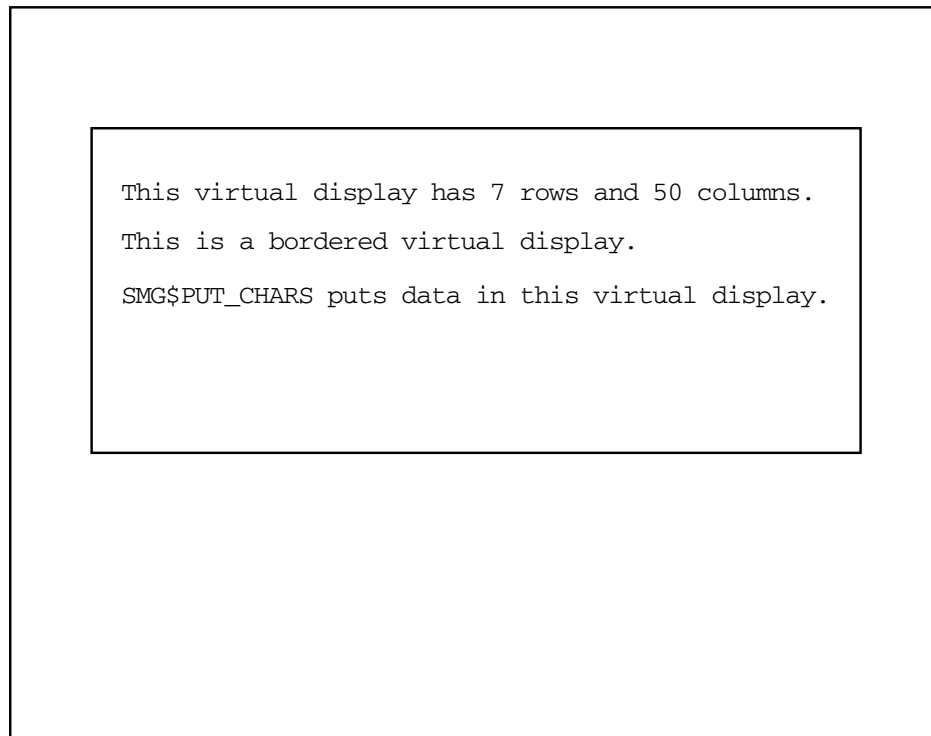
 STATUS = SMG$ERASE_LINE (DISPLAY1, 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$ERASE_LINE (DISPLAY1, 4, 9)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 END
```

The initial output generated by the FORTRAN program is shown in Figure SMG-24.

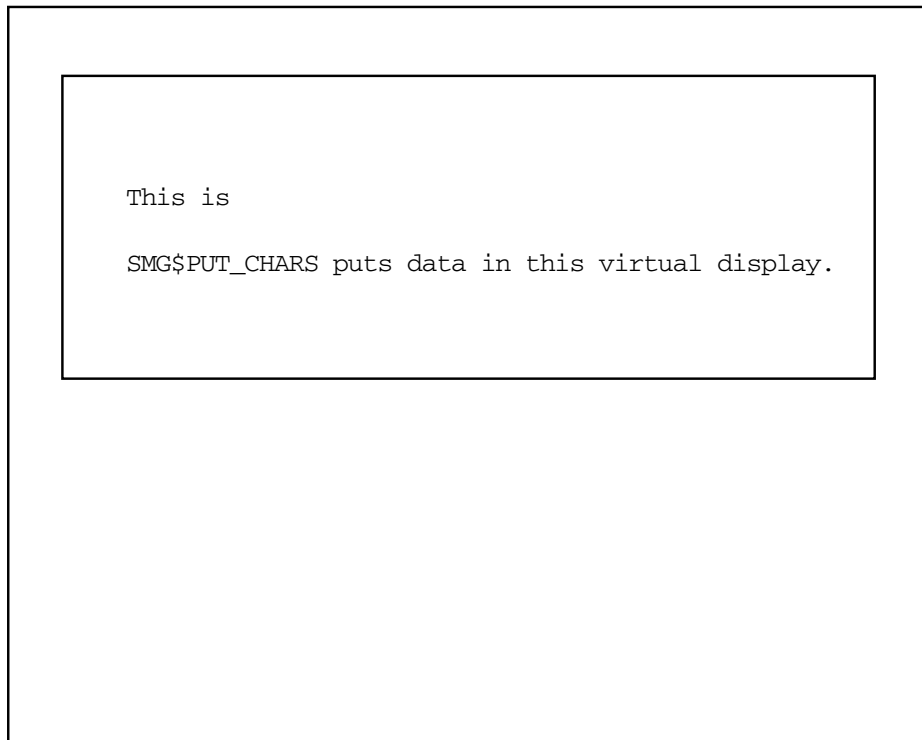
**Figure SMG-24 Initial Output Generated by FORTRAN Program Calling SMG\$ERASE\_LINE**



ZK-4108-GE

The output generated after the call to SMG\$ERASE\_LINE is shown in Figure SMG-25.

**Figure SMG-25 Output Generated After the Call to SMG\$ERASE\_LINE**



ZK-4117-GE

## SMG\$ERASE\_PASTEBOARD

---

### SMG\$ERASE\_PASTEBOARD—Erase Pasteboard

The Erase Pasteboard routine erases the contents of a pasteboard.

#### Format

SMG\$ERASE\_PASTEBOARD pasteboard-id

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

|                      |                     |
|----------------------|---------------------|
| <b>pasteboard-id</b> |                     |
| OpenVMS usage        | identifier          |
| type                 | longword (unsigned) |
| access               | read only           |
| mechanism            | by reference        |

Specifies the pasteboard to be erased. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

#### Description

SMG\$ERASE\_PASTEBOARD erases the contents of a specified pasteboard. The physical cursor is left at position 1,1. If there are any virtual displays pasted to the pasteboard, they will be redrawn the next time the Screen Management Facility is used to output to the pasteboard.

#### Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_BATWAS_ON | Pasteboard is batched.         |
| SMG\$_WRONUMARG | Wrong number of arguments.     |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SS\$_xxxx       | Any status from \$QIOW.        |

#### Example

```
C+
C This DEC Fortran example program demonstrates the use of
C SMG$ERASE_PASTEBOARD.
C-

 IMPLICIT INTEGER*4 (A-Z)
 CHARACTER*80 OUT_STR,TRIM_STR
 CHARACTER*18 PROMPT //'Please enter data '/
```



## SMG\$ERASE\_PASTEBOARD

```
 SMG$M_BOLD = 1
 SMG$M_REVERSE = 2
 SMG$M_BLINK = 4
 SMG$M_UNDERLINE = 8
C+
C Establish the terminal keyboard as the virtual keyboard
C by calling SMG$CREATE_VIRTUAL_KEYBOARD.
C-
 STATUS = SMG$CREATE_VIRTUAL_KEYBOARD(KEYBOARD_ID,,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Establish the terminal screen as a pasteboard using
C SMG$CREATE_PASTEBOARD.
C-
 STATUS = SMG$CREATE_PASTEBOARD (NEW_PID,,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Establish a virtual display region by
C calling SMG$CREATE_VIRTUAL_DISPLAY.
C-
 STATUS = SMG$CREATE_VIRTUAL_DISPLAY (5,80,DISPLAY_ID,,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Paste the virtual display to the screen, starting at
C row 10, column 15. To paste the virtual display, use
C SMG$PASTE_VIRTUAL_DISPLAY.
C-
 STATUS = SMG$PASTE_VIRTUAL_DISPLAY(DISPLAY_ID,NEW_PID,10,15)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Prompt the user for input, and accept that input using
C SMG$READ_STRING.
C-
 STATUS = SMG$READ_STRING(KEYBOARD_ID,OUT_STR,PROMPT,,,,,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Clear the screen using SMG$ERASE_PASTEBOARD.
C-
 STATUS = SMG$ERASE_PASTEBOARD (NEW_PID)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Trim any trailing blanks from the user input
C by calling STR$TRIM.
C-
 STATUS = STR$TRIM(TRIM_STR,OUT_STR,STR_LEN)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Display the data input by the user using SMG$PUT_CHARS
C and SMG$PUT_LINE.
C-
 STATUS = SMG$PUT_CHARS(DISPLAY_ID,'You entered: ',,,,,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
 STATUS = SMG$PUT_LINE(DISPLAY_ID,TRIM_STR(1:STR_LEN),,
1 SMG$M_REVERSE,0,,)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
 END
```

This FORTRAN program calls Run-Time Library Screen Management routines to format screen output, and to accept and display user input.

## SMG\$EXECUTE\_COMMAND

---

### SMG\$EXECUTE\_COMMAND—Execute Command in a Subprocess

The Execute Command in a Subprocess routine executes the specified command in the subprocess created with the SMG\$CREATE\_SUBPROCESS routine.

#### Format

SMG\$EXECUTE\_COMMAND display-id ,command-desc [,flags] [,ret-status]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Display identifier of the virtual display with which the subprocess is associated. The **display-id** argument is the address of an unsigned longword containing this identifier.

##### command-desc

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Command string. The **command-desc** argument is the address of a descriptor pointing to the command string.

##### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask that specifies optional behavior. The **flags** argument is the address of an unsigned longword that contains the flag. The valid values for **flags** are as follows:

|                     |                                                                                                                                                                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_DATA_FOLLOWS | Input data follows. The next call to SMG\$EXECUTE_COMMAND contains input data for the currently executing command. Do not specify this value if this is the last input data item. If you do specify this value, <b>ret-status</b> is not returned. |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

SMG\$M\_SEND\_EOF                      Send end-of-file marker. The end-of-file marker is sent to the subprocess.

**ret-status**

OpenVMS usage    cond\_value  
 type            longword (unsigned)  
 access           write only  
 mechanism        by reference

Optional status of the executed command, provided that the commands are not being buffered. The **ret-status** argument is the address of an unsigned longword containing this status.

## Description

SMG\$EXECUTE\_COMMAND lets you execute the specified command in the subprocess created with SMG\$CREATE\_SUBPROCESS. If commands are being buffered, this routine returns control after the command has been buffered, and the user-specified AST routine is invoked when the command completes. If commands are not being buffered, SMG\$EXECUTE\_COMMAND waits until the command has completed execution before returning the status of the command.

When specifying the command string, you must specify a dollar sign (\$) as the first character of any DCL command. Any command string that does not begin with a dollar sign is assumed to be input data for the previous command. SMG\$EXECUTE\_COMMAND outputs the commands and their output to the specified virtual display as they are executed. Do not perform I/O to the specified virtual display. Note that the commands SPAWN, GOTO, and LOGOUT are illegal to use as command strings and generate unpredictable results.

Since I/O is performed using mailboxes and not through the terminal driver, command prompts and single-character commands such as Ctrl/C, Ctrl/Y, Ctrl/Z, and so forth have no effect. You should specify SMG\$M\_SEND\_EOF for the **flags** parameter in order to send a Ctrl/Z to the subprocess.

## Condition Values Returned

|                 |                                              |
|-----------------|----------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                  |
| SMG\$_INPTOOLON | Input is longer than 255 characters.         |
| SMG\$_NOSUBEXI  | No subprocess exists.                        |
| SS\$_xxxx       | Any status from \$QIO, \$DCLAST, or \$SYNCH. |
| LIB\$_xxxx      | Any status from LIB\$ANALYZE_SDESC.          |
| SMG\$_xxxx      | Any status from SMG\$PUT_LINE.               |

---

### SMG\$FIND\_CURSOR\_DISPLAY—Find Display that Contains the Cursor

The Find Display that Contains the Cursor routine returns the identifier of the most recently pasted virtual display that contains the physical cursor.

#### Format

SMG\$FIND\_CURSOR\_DISPLAY pasteboard-id ,display-id [,pasteboard-row] [,pasteboard-column]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### pasteboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard in which the physical cursor is to be found. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Receives the identifier of the display in which the physical cursor was found. The **display-id** argument is the address of an unsigned longword into which the display identifier is written.

##### pasteboard-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

The row position at which to begin the search for the physical cursor. The optional **pasteboard-row** argument is the address of a signed longword containing the pasteboard row. You can use **pasteboard-row** instead of the physical cursor row.

##### pasteboard-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |

mechanism            by reference

The column position at which to begin the search for the physical cursor. The optional **pasteboard-column** argument is the address of a signed longword containing the pasteboard column. You can use **pasteboard-column** instead of the physical cursor column.

## Description

SMG\$FIND\_CURSOR\_DISPLAY determines which virtual display contains the physical cursor on a specified pasteboard, and returns the virtual display's identifier. SMG\$FIND\_CURSOR\_DISPLAY returns the **display-id** of the most recently pasted virtual display that contains the physical cursor. If no virtual display contains the physical cursor, this routine returns a zero, which is an invalid display identifier.

## Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SMG\$_WRONUMARG | Wrong number of arguments.     |

## SMG\$FLUSH\_BUFFER

---

### SMG\$FLUSH\_BUFFER—Flush Buffer

The Flush Buffer routine flushes all buffered output to the terminal.

#### Format

SMG\$FLUSH\_BUFFER pasteboard-id

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

|                      |                     |
|----------------------|---------------------|
| <b>pasteboard-id</b> |                     |
| OpenVMS usage        | identifier          |
| type                 | longword (unsigned) |
| access               | read only           |
| mechanism            | by reference        |

Specifies the pasteboard to be flushed. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

#### Description

SMG\$FLUSH\_BUFFER causes all buffered output that is not already output to be sent to the pasteboard immediately. The Screen Management Facility outputs the text when the buffer is full; therefore, this routine is only needed when a partial buffer must be output. The calling program would normally call this routine just before performing some CPU-intensive calculations, or whenever the pasteboard must be up to date.

#### Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_WRONUMARG | Wrong number of arguments.     |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SS\$_xxxx       | Any error from \$QIOW.         |

---

## SMG\$GET\_BROADCAST\_MESSAGE—Get Broadcast Message

The Get Broadcast Message routine determines if a message has been broadcast to the pasteboard and returns the message.

### Format

SMG\$GET\_BROADCAST\_MESSAGE pasteboard-id [,message] [,message-length] [,message-type]

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### pasteboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard to be checked for the presence of a broadcast message. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

#### message

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

A string that receives the broadcast message, if such a message is available. The **message** argument is the address of a descriptor that points to the string into which the message text is written. If this argument is omitted, the broadcast message is discarded.

#### message-length

|               |                 |
|---------------|-----------------|
| OpenVMS usage | word_unsigned   |
| type          | word (unsigned) |
| access        | write only      |
| mechanism     | by reference    |

Receives the actual length of the broadcast message. The **message-length** argument is the address of an unsigned word into which is written the length of the message.

#### message-type

|               |                 |
|---------------|-----------------|
| OpenVMS usage | word_unsigned   |
| type          | word (unsigned) |
| access        | write only      |
| mechanism     | by reference    |

## SMG\$GET\_BROADCAST\_MESSAGE

Receives the type of broadcast message. The **message-type** argument is the address of an unsigned word into which is written the type of message. Values for **message-type** are defined by the \$MSGDEF library definition. If the value for **message-type** is not MSG\$\_TRMBRDCST, the condition value returned is SMG\$\_NOBRDMSG.

### Description

SMG\$GET\_BROADCAST\_MESSAGE determines if any broadcast messages have been sent to the specified pasteboard while broadcast trapping was enabled and, if so, returns the message in the **message** argument. You may call this routine repeatedly until all broadcast messages have been returned. If there are no more broadcast messages available, SMG\$GET\_BROADCAST\_MESSAGE returns the success status SMG\$\_NO\_MORMSG.

### Condition Values Returned

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                           |
| SMG\$_NO_MORMSG | Successful completion. No more messages to be returned. |
| SMG\$_NONBRDMSG | Nonbroadcast message returned.                          |
| SMG\$_WRONUMARG | Wrong number of arguments.                              |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> .                          |

Any condition values returned by LIB\$SCOPY\_DXDX.



---

## SMG\$GET\_DISPLAY\_ATTR—Get Display Attributes

The Get Display Attributes routine receives the attributes associated with a virtual display.

### Format

```
SMG$GET_DISPLAY_ATTR display-id [,height] [,width] [,display-attributes] [,video-attributes]
 [,character-set] [,flags]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Specifies the virtual display for which information is requested. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### height

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |

Receives the number of rows in the display. The optional **height** argument is the address of a signed longword into which the height is written.

## SMG\$GET\_DISPLAY\_ATTR

### **width**

OpenVMS usage longword\_signed  
type longword (signed)  
access write only  
mechanism by reference

Receives the number of columns in the display. The optional **width** argument is the address of a signed longword into which is written the number of columns in the display.

### **display-attributes**

OpenVMS usage mask\_longword  
type longword (unsigned)  
access write only  
mechanism by reference

Receives the current default display attributes. The optional **display-attributes** argument is the address of an unsigned longword into which the current display attributes are written.

Valid values for **display-attributes** are as follows:

|                         |                                                                                                                                                     |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BORDER           | Indicates a bordered display.                                                                                                                       |
| SMG\$M_BLOCK_BORDER     | Indicates a block bordered display.                                                                                                                 |
| SMG\$M_DISPLAY_CONTROLS | Indicates that control characters such as carriage return and line feed are displayed as graphic characters.                                        |
| SMG\$M_TRUNC_ICON       | Indicates that an icon (generally a diamond shape) is displayed where truncation of a line exceeding the width of the virtual display has occurred. |

### **video-attributes**

OpenVMS usage mask\_longword  
type longword (unsigned)  
access write only  
mechanism by reference

Receives the current default video attributes. The optional **video-attributes** argument is the address of an unsigned longword into which the current video attributes are written.

Valid video attributes are as follows:

|                  |                                                                                                                           |
|------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK     | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD      | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE   | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |

SMG\$M\_USER1 through SMG\$M\_USER8      Displays user-defined attributes.

#### character-set

OpenVMS usage    longword\_unsigned  
 type            longword (unsigned)  
 access           write only  
 mechanism       by reference

Receives the default character set for all text in this virtual display. The optional **character-set** argument is the address of an unsigned longword that specifies the character set. If the character set is undefined, receives SMG\$C\_UNKNOWN.

| Value                | Character Set         |
|----------------------|-----------------------|
| SMG\$C_ASCII         | ASCII                 |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics  |
| SMG\$C_HANYU         | DEC Hanyu             |
| SMG\$C_UNKNOWN       | Unknown character set |

#### flags

OpenVMS usage    mask\_longword  
 type            longword (unsigned)  
 access           write only  
 mechanism       by reference

Optional bit mask specifying attributes of the specified display. The **flags** argument is the address of an unsigned longword containing the flag. Valid values for **flags** are as follows:

SMG\$M\_SUBPROCESS      Display has a subprocess attached to it.  
 SMG\$M\_MENU            Display contains a menu.  
 SMG\$M\_VIEWPORT       Display contains a viewport.

## Description

SMG\$GET\_DISPLAY\_ATTR receives the attributes of a virtual display.

Original SMG routines support only ASCII(default) and DEC Special Graphics, and SMG\$GET\_DISPLAY\_ATTR returns either SMG\$C\_ASCII or SMG\$C\_SPEC\_GRAPHICS in the **character-set** argument. But Chinese SMG supports additional character set for the Chinese language, and SMG\$GET\_DISPLAY\_ATTR returns some enhanced values in the argument.

## Condition Values Returned

SS\$\_NORMAL            Normal successful completion.  
 SMG\$\_INVDIS\_ID       Invalid **display-id**.  
 SMG\$\_WRONUMARG       Wrong number of arguments.

## SMG\$GET\_KEY\_DEF

---

### SMG\$GET\_KEY\_DEF—Get Key Definition

The Get Key Definition routine returns the key definition for a specified key.

#### Format

```
SMG$GET_KEY_DEF key-table-id ,key-name [,if-state] [,attributes] [,equivalence-string] [,state-string]
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **key-table-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the key table from which you are extracting a definition. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

**Key-table-id** is returned by SMG\$CREATE\_KEY\_TABLE.

##### **key-name**

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Specifies the name of the key associated with the definition. The **key-name** argument is the address of a descriptor pointing to the key name.

Table 3–1 lists the valid key names.

##### **if-state**

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Specifies the current state name in effect after the key is pressed. The **if-state** argument is the address of a descriptor pointing to the state name.

See SMG\$ADD\_KEY\_DEF for more information.

##### **attributes**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Receives the attributes bit mask for this key definition. The **attributes** argument is the address of a longword into which is written the bit mask describing the key's attributes.

Valid values are as follows:

|                      |                                                                                                                                                                                                                                                                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_KEY_NOECHO    | If set, this bit specifies that <b>equiv_string</b> is not to be echoed when this key is pressed. If clear, <b>equiv_string</b> is echoed. If SMG\$M_KEY_TERMINATE is not set, SMG\$M_KEY_NOECHO is ignored.                                                                                                                                      |
| SMG\$M_KEY_TERMINATE | If set, this bit specifies that when this key is pressed (as qualified by <b>if-state</b> ), the input line is complete and more characters should not be accepted. If clear, more characters may be accepted.                                                                                                                                    |
| SMG\$M_KEY_LOCK      | If set, and if <b>state-string</b> is specified, the state name specified by <b>state-string</b> remains the current state until explicitly changed by a subsequent keystroke whose definition includes a <b>state-string</b> . If clear, the state name specified by <b>state-string</b> remains in effect only for the next defined key stroke. |
| SMG\$M_KEY_PROTECTED | If set, this bit specifies that this key definition cannot be modified or deleted. If clear, the key definition can be modified or deleted.                                                                                                                                                                                                       |

#### **equivalence-string**

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

Receives the equivalence string for this key definition. The **equivalence-string** argument is the address of a descriptor pointing to the string into which is written the equivalence string.

#### **state-string**

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

Receives the new state name, if any, which is set by this key definition. The **state-string** argument is the address of a descriptor pointing to the string into which is written the new state string.

## **Description**

SMG\$GET\_KEY\_DEF returns the key definition associated with a specified **key-name** and **if-state**. This key definition may be used in calls to SMG\$READ\_COMPOSED\_LINE.

## SMG\$GET\_KEY\_DEF

### Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVKEYNAM | Invalid <b>key-name</b> .     |
| SMG\$_INVKTB_ID | Invalid <b>key-table-id</b> . |
| SMG\$_KEYNOTDEF | Key not defined.              |
| SMG\$_WRONUMARG | Wrong number of arguments.    |

Any condition values returned by LIB\$SCOPY\_DXDX.

---

## SMG\$GET\_KEYBOARD\_ATTRIBUTES—Get Keyboard Attributes

The Get Keyboard Attributes routine gets information about a virtual keyboard and leaves it in a user-supplied area: the keyboard information table (KIT).

### Format

SMG\$GET\_KEYBOARD\_ATTRIBUTES keyboard-id ,keyboard-info-table ,keyboard-info-table-size

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### keyboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Keyboard identifier. The **keyboard-id** argument is the address of an unsigned longword containing the identifier of the virtual keyboard from which to read.

You create a virtual keyboard by calling the SMG\$CREATE\_VIRTUAL\_KEYBOARD routine.

#### keyboard-info-table

|               |                               |
|---------------|-------------------------------|
| OpenVMS usage | unspecified                   |
| type          | unspecified                   |
| access        | write only                    |
| mechanism     | by reference, array reference |

Receives the keyboard attributes. The **keyboard-info-table** argument is the address of a data block into which the keyboard attributes are written.

The keyboard information table (KIT) is a byte block whose size and field references are described in \$SMGDEF. It is the caller's responsibility to allocate the correct size block and to pass its address to this routine.

The values in the **keyboard-info-table** can be accessed through the following symbolic names:

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| SMG\$L_DEV_CHAR    | Device characteristics (longword).                     |
| SMG\$L_DEV_DEPEND  | Specific characteristics 1 (longword).                 |
| SMG\$L_DEV_DEPEND2 | Specific characteristics 2 (longword).                 |
| SMG\$B_DEV_CLASS   | Device class (byte) — for example, DC\$_TERM.          |
| SMG\$B_RECALL_NUM  | Size of recall buffer (byte). *                        |
| SMG\$B_DEVTYPE     | Physical device type (byte) — for example, TT\$_VT100. |





## SMG\$GET\_KEYBOARD\_ATTRIBUTES

```
PRINT SMG_INFO::SMG$L_DEV_CHAR ! Device characteristics
PRINT SMG_INFO::SMG$L_DEV_DEPEND ! Specific characteristics (1)
PRINT SMG_INFO::SMG$L_DEV_DEPEND2 ! Specific characteristics (2)
PRINT SMG_INFO::SMG$B_DEV_CLASS ! Device class (DC$TERM)
PRINT SMG_INFO::SMG$B_RECALL_NUM ! Size of SMG recall buffer
PRINT SMG_INFO::SMG$B_DEV_TYPE ! Device type (DT$VT100)
PRINT SMG_INFO::SMG$B_TYPEAHD_CHAR ! First character in
 ! typeahead buffer
PRINT SMG_INFO::SMG$W_NUM_COLUMNS ! Terminal width
PRINT SMG_INFO::SMG$W_TYPEAHD_CNT ! Number of characters in
 ! typeahead buffer

END
```

## SMG\$GET\_NUMERIC\_DATA

---

### SMG\$GET\_NUMERIC\_DATA—Get Numeric Terminal Data

The Get Numeric Terminal Data routine accesses `TERMTABLE.EXE` and returns the value associated with a specified Boolean or numeric capability.

#### Format

```
SMG$GET_NUMERIC_DATA termtable-address ,request-code ,buffer-address
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **termtable-address**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | address             |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the address of the `TERMTABLE` entry for the desired terminal. The **termtable-address** argument is the address of an unsigned longword that contains the address of the terminal capabilities table (`TERMTABLE`).

Before calling `SMG$GET_NUMERIC_DATA`, you must obtain this terminal table address by calling either `SMG$INIT_TERM_TABLE` or `SMG$INIT_TERM_TABLE_BY_TYPE`.

##### **request-code**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Request code that specifies the desired capability. The **request-code** argument is an unsigned longword constant containing this request code. The request code is of the form `SMG$K_code` where code corresponds to a keyword in the terminal capabilities table (`TERMTABLE`), for example, `ANSI_CRT`. The `SMG$K_code` constants can be found in the Digital-supplied library `$SMGTRMPTR`.

See Table 5-1, Table 5-2, Table 5-3, and Table 5-4 in Part I of this manual for valid capability fields.

##### **buffer-address**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | address             |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Address of the first byte of the longword to which `SMG$GET_NUMERIC_DATA` writes the numeric capability data. The **buffer-address** argument is an unsigned longword that contains the address of this buffer.

## Description

SMG\$GET\_NUMERIC\_DATA extracts the requested numeric information from a specified terminal table. Before calling SMG\$GET\_NUMERIC\_DATA, you must obtain that terminal table address by calling either SMG\$INIT\_TERM\_TABLE or SMG\$INIT\_TERM\_TABLE\_BY\_TYPE. This routine need only be used if you are doing your own TERMTABLE access, and only when you perform direct (non-SMG\$) I/O to terminals.

## Condition Values Returned

|                 |                                 |
|-----------------|---------------------------------|
| SS\$_NORMAL     | Normal successful completion.   |
| SMG\$_INVREQCOD | Invalid request code.           |
| SMG\$_INVTERTAB | Invalid terminal table address. |

## SMG\$GET\_PASTEBOARD\_ATTRIBUTES

---

### SMG\$GET\_PASTEBOARD\_ATTRIBUTES—Get Pasteboard Attributes

The Get Pasteboard Attributes routine gets pasteboard attributes and stores them in the pasteboard information table.

#### Format

```
SMG$GET_PASTEBOARD_ATTRIBUTES pasteboard-id ,pasteboard-info-table
 ,pasteboard-info-table-size
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **pasteboard-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard for which information is requested. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

##### **pasteboard-info-table**

|               |                               |
|---------------|-------------------------------|
| OpenVMS usage | unspecified                   |
| type          | unspecified                   |
| access        | write only                    |
| mechanism     | by reference, array reference |

Receives the pasteboard attributes. The **pasteboard-info-table** argument is the address of a data structure into which are written the pasteboard attributes.

The values in the **pasteboard-info-table** can be accessed through the following symbolic names:

|                   |                                             |
|-------------------|---------------------------------------------|
| SMG\$L_DEVCHAR    | Device characteristics (longword).          |
| SMG\$L_DEVDEPEND  | Specific characteristics 1 (longword).      |
| SMG\$L_DEVDEPEND2 | Specific characteristics 2 (longword).      |
| SMG\$B_DEVCLASS   | Device class (byte)—for example, DC\$_TERM. |

## SMG\$GET\_PASTEBOARD\_ATTRIBUTES

|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SMG\$B_SMG_DEVTYPE     | Internal SMG device type (byte). The four possible values for SMG\$B_SMG_DEVTYPE are as follows:<br><br>SMG\$K_UNKNOWN<br>SMG\$K_VTForeign<br>SMG\$K_HARDCOPY<br>SMG\$K_VTTERMTABLE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| SMG\$B_PHY_DEVTYPE     | Physical device type (byte)—for example, TT\$_VT100. The possible values for SMG\$B_PHY_DEVTYPE are defined in \$TTDEF in STARLET.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| SMG\$B_ROWS            | Number of rows on pasteboard (byte).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| SMG\$W_WIDTH           | Pasteboard width (word).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| SMG\$B_COLOR           | Background color setting (byte). Valid values for SMG\$B_COLOR are as follows:<br><br>SMG\$C_COLOR_UNKNOWN           Unknown background color<br>SMG\$C_COLOR_WHITE           White background<br>SMG\$C_COLOR_BLACK           Black background<br>SMG\$C_COLOR_BLUE           Blue background<br>SMG\$C_COLOR_CYAN           Cyan (green-blue) background<br><br>SMG\$C_COLOR_GREEN           Green background<br>SMG\$C_COLOR_MAGENTA       Magenta background<br>SMG\$C_COLOR_RED           Red background<br>SMG\$C_COLOR_YELLOW       Yellow background<br>SMG\$C_COLOR_LIGHT       Light background<br>SMG\$C_COLOR_DARK       Dark background<br>SMG\$C_COLOR_USER1       User-defined background 1<br><br>SMG\$C_COLOR_USER2       User-defined background 2 |
| SMG\$B_PARITY          | Parity attributes (byte)—this field is zero if the pasteboard is not a terminal.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| SMG\$W_SPEED           | Terminal speed (word)—this field is zero if the pasteboard is not a terminal.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| SMG\$W_FILL            | Fill characteristics (word)—this field is zero if the pasteboard is not a terminal.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| SMG\$W_PHYS_CURSOR_ROW | Pasteboard row containing physical cursor (word).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| SMG\$W_PHYS_CURSOR_COL | Pasteboard column containing physical cursor (word).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| SMG\$L_CURSOR_DID      | Display identifier of topmost display containing physical cursor (longword).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

### **pasteboard-info-table-size**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |

## SMG\$GET\_PASTEBOARD\_ATTRIBUTES

mechanism            by reference

Specifies the number of bytes in the pasteboard information table. The **pasteboard-info-table-size** argument is the address of an unsigned longword that contains the size (in bytes) of the pasteboard information table.

The size you specify must be exact. You can specify this size with the symbolic constant SMG\$C\_PASTEBOARD\_INFO\_BLOCK.

### Description

SMG\$GET\_PASTEBOARD\_ATTRIBUTES gets pasteboard attributes and stores them in the pasteboard information table.

### Condition Values Returned

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                   |
| SMG\$_INVARG    | Incorrect size specified in <b>pasteboard-info-table-size</b> . |
| SMG\$_WRONUMARG | Wrong number of arguments.                                      |

### Example

```
10 !+
 !This VAX BASIC example demonstrates the use of the routine
 ! SMG$GET_PASTEBOARD_ATTRIBUTES.
 !-

 OPTION TYPE = EXPLICIT
 OPTION CONSTANT TYPE = INTEGER

 %INCLUDE "$SMGDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"
 %INCLUDE "$SSDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"

 EXTERNAL LONG FUNCTION LIB$SIGNAL(LONG BY VALUE), &
 SMG$CREATE_PASTEBOARD(LONG), &
 SMG$GET_PASTEBOARD_ATTRIBUTES(LONG, ANY, LONG)

 DECLARE SMG$ATTRIBUTE_INFO_BLOCK SMG_INFO
 DECLARE LONG S, PASTEBOARD_ID

 S = SMG$CREATE_PASTEBOARD(PASTEBOARD_ID)
 IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL(S) END IF

 S = SMG$GET_PASTEBOARD_ATTRIBUTES(PASTEBOARD_ID, &
 SMG_INFO, &
 SMG$C_PASTEBOARD_INFO_BLOCK)
 IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL(S) END IF
```

## SMG\$GET\_PASTEBOARD\_ATTRIBUTES

```
PRINT SMG_INFO::SMG$L_DEV_CHAR ! Device characteristics
PRINT SMG_INFO::SMG$L_DEV_DEPEND ! Specific characteristics (1)
PRINT SMG_INFO::SMG$L_DEV_DEPEND2 ! Specific characteristics (2)
PRINT SMG_INFO::SMG$B_DEV_CLASS ! Device class (DC$_TERM)
PRINT SMG_INFO::SMG$B_PBD_TYPE ! SMG type (SMG$K_VTTERMTABLE)
PRINT SMG_INFO::SMG$B_DEV_TYPE ! Device type (DT$_VT100)
PRINT SMG_INFO::SMG$B_NUM_ROWS ! Number of rows on pasteboard
PRINT SMG_INFO::SMG$W_NUM_COLUMNS ! Number of cols on pasteboard
PRINT SMG_INFO::SMG$B_PBD_COLOR ! Pasteboard background color
PRINT SMG_INFO::SMG$B_DEV_PARITY ! Device parity characteristic
PRINT SMG_INFO::SMG$W_DEV_SPEED ! Device speed characteristic
PRINT SMG_INFO::SMG$W_DEV_FILL ! Device fill characteristic
PRINT SMG_INFO::SMG$W_PHYS_CURSOR_ROW ! Physical cursor row
PRINT SMG_INFO::SMG$W_PHYS_CURSOR_COL ! Physical cursor column
PRINT SMG_INFO::SMG$L_DISPLAY_ID ! Display_id containing cursor
END
```

---

## SMG\$GET\_PASTING\_INFO—Return Pasting Information

Provided that the specified virtual display is currently pasted, the Return Pasting Information routine returns the row and column of the pasting.

### Format

SMG\$GET\_PASTING\_INFO display-id ,pasteboard-id ,flags [,pasteboard-row] [,pasteboard-column]

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the virtual display to be examined. The **display-id** argument is the address of an unsigned longword containing the identifier of this virtual display.

#### pasteboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the pasteboard on which the virtual display is pasted. The **pasteboard-id** argument is the address of an unsigned longword containing the identifier of this pasteboard.

#### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Bit mask indicating the status of the specified virtual display with respect to the specified pasteboard. The **flags** argument is the address of an unsigned longword that contains the flag. Valid values for **flags** are as follows:

|                       |                                                                                                                                |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------|
| 0                     | The virtual display is not pasted to the specified pasteboard.                                                                 |
| SMG\$M_DISPLAY_PASTED | The virtual display specified by <b>display-id</b> is pasted to the pasteboard specified by the <b>pasteboard-id</b> argument. |



**pasteboard-row**

OpenVMS usage longword\_signed  
 type longword (signed)  
 access write only  
 mechanism by reference

Row of the pasteboard that contains row 1 of the specified virtual display. The optional **pasteboard-row** argument is the address of a signed longword containing the number of the pasteboard row that contains the first row of the virtual display.

**pasteboard-column**

OpenVMS usage longword\_signed  
 type longword (signed)  
 access write only  
 mechanism by reference

Column of the pasteboard that contains column 1 of the specified virtual display. The optional **pasteboard-column** argument is the address of a signed longword containing the number of the pasteboard column that contains the first column of the virtual display.

**Description**

SMG\$GET\_PASTING\_INFO first checks to see if the virtual display specified by **display-id** is pasted to the pasteboard specified by **pasteboard-id**. If this virtual display is pasted to this pasteboard, SMG\$GET\_PASTING\_INFO returns the row and column numbers of the pasteboard that correspond to row 1 and column 1 of the pasted virtual display.

**Condition Values Returned**

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .    |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SMG\$_WRONUMARG | Wrong number of arguments.     |
| SMG\$_ILLBATFNC | Display is batched.            |

---

## SMG\$GET\_TERM\_DATA—Get Terminal Data

The Get Terminal Data routine accesses TERMTABLE.EXE and returns the character sequence that causes a terminal to perform a specified operation.

### Format

```
SMG$GET_TERM_DATA termtable-address ,request-code ,maximum-buffer-length ,return-length
 ,capability-data [,input-argument-vector]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### termtable-address

|               |                     |
|---------------|---------------------|
| OpenVMS usage | address             |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the address of the TERMTABLE entry for the desired terminal. The **termtable-address** argument is the address of an unsigned longword that contains the address of the terminal capabilities table (TERMTABLE).

**Termtable-address** is returned by SMG\$INIT\_TERM\_TABLE or SMG\$INIT\_TERM\_TABLE\_BY\_TYPE.

#### request-code

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Longword constant of the form SMG\$K\_code, where **code** is the name of the desired capability field. The **request-code** argument is the address of an unsigned longword that contains the request code. The SMG\$K\_code constants can be found in the Digital-supplied library \$SMGTRMPTR.

See Table 5–1, Table 5–2, Table 5–3, and Table 5–4 in Part I of this manual for valid capability fields.

#### maximum-buffer-length

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Maximum length of the buffer into which the requested capability data is written. The **maximum-buffer-length** argument is the address of a signed longword that contains the maximum number of bytes that can be written into the buffer.

**return-length**

OpenVMS usage longword\_signed  
 type longword (signed)  
 access write only  
 mechanism by reference

Receives the number of bytes actually written into the buffer. The **return-length** argument is the address of a signed longword into which is written the number of bytes transferred into the buffer.

**buffer-address**

OpenVMS usage address  
 type longword (unsigned)  
 access write only  
 mechanism by reference

Address of the first byte of the buffer which is to receive the capability data. The **buffer-address** argument is an unsigned longword that contains the address of the buffer.

**input-argument-vector**

OpenVMS usage vector\_longword\_unsigned  
 type longword (unsigned)  
 access read only  
 mechanism by reference, array reference

Address of a list of longwords used for capabilities that require a variable number of arguments, and for those that require substitution or arithmetic operations on an argument. The **input-argument-vector** argument is the address of an array of unsigned longwords that contains capability arguments. The first longword must contain the number of arguments that follow.

**Description**

SMG\$GET\_TERM\_DATA should be used only when you perform direct (non-SMG\$) I/O to terminals. It accesses the TERMTABLE.EXE entry for the specified type of terminal and returns the character sequence that performs the specified operation. It is up to you to send this character sequence to the terminal.

**Condition Values Returned**

|                 |                                 |
|-----------------|---------------------------------|
| SS\$_NORMAL     | Normal successful completion.   |
| SMG\$_INVERTAB  | Invalid terminal table address. |
| SMG\$_INVREQCOD | Invalid request code.           |

---

## SMG\$GET\_VIEWPORT\_CHAR—Get Characteristics of Display Viewport

The Get Characteristics of Display Viewport routine returns the characteristics of the specified viewport.

### Format

```
SMG$GET_VIEWPORT_CHAR display-id [,viewport-row-start] [,viewport-column-start]
 [,viewport-number-rows] [,viewport-number-columns]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the virtual display associated with the viewport. The **display-id** argument is the address of an unsigned longword containing this identifier.

#### viewport-row-start

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |

Optional argument that receives the starting row number of the viewport. The **viewport-row-start** argument is the address of a signed longword that receives this row number.

#### viewport-column-start

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |

Optional argument that receives the starting column number of the specified viewport. The **viewport-column-start** argument is the address of a signed longword that receives this column number.

#### viewport-number-rows

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |

Optional argument that receives the number of rows in the specified viewport. The **viewport-number-rows** argument is the address of a signed longword that receives this number.

#### viewport-number-columns

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |

Optional argument that receives the number of columns in the specified viewport. The **viewport-number-columns** argument is the address of a signed longword that receives this number.

## Description

SMG\$GET\_VIEWPORT\_CHAR returns the requested characteristics of the specified viewport.

## Condition Values Returned

|                   |                                                  |
|-------------------|--------------------------------------------------|
| SS\$_NORMAL       | Normal successful completion.                    |
| SMG\$_INVDIS_ID   | Invalid <b>display-id</b> .                      |
| SMG\$_NO_WINASSOC | No viewport associated with the virtual display. |
| SMG\$_WRONUMARG   | Wrong number of arguments.                       |

## Example

```

C+
C This DEC Fortran example demonstrates the use of SMG$GET_VIEWPORT_CHAR.
C The viewport created will start at row 3, column 4. It will consist of
C 7 rows and 29 columns. Note the parameters used in the SMG$CREATE_VIEWPORT
C routine. I request 26 rows and 55 columns, but my viewport is truncated
C to fit.
C-
 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'

C Create the virtual display. Give it a border.
 ROWS = 4
 COLUMNS = 34
 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$_M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%val(STATUS))

C Create the pasteboard.
 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%val(STATUS))

C Put data in the virtual display.
 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 'This is row number 1 of 4, you see', 1, 1)
 IF (.not. STATUS) CALL LIB$SIGNAL(%val(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 'This is row number 2 of 4, you see', 2, 1)
 IF (.not. STATUS) CALL LIB$SIGNAL(%val(STATUS))

```

## SMG\$GET\_VIEWPORT\_CHAR

```
STATUS = SMG$PUT_CHARS (DISPLAY1,
1 'This is row number 3 of 4, you see', 3, 1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS (DISPLAY1,
1 'This is row number 4 of 4, you see', 4, 1)
IF (.not. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C Paste the virtual display.

STATUS = SMG$COPY_VIRTUAL_DISPLAY(DISPLAY1,DISPLAY2)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$LABEL_BORDER (DISPLAY1, 'Full Display',,,SMG$M_BOLD)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$LABEL_BORDER (DISPLAY2,'Viewport',,,SMG$M_BOLD)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 2, 2)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$CREATE_VIEWPORT (DISPLAY2, 1, 5, 26, 55)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY2, PASTE1, 8, 2)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

CALL SMG$SET_PHYSICAL_CURSOR(PASTE1, 16, 1)
TYPE *, ' '
TYPE *, LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$GET_VIEWPORT_CHAR (DISPLAY2, A, B, C, D)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

TYPE *, ' '
WRITE(5,7) A,B
7 FORMAT(1X,'Row start = ',I2,8X,'Column start = ',I2)
TYPE *, ' '
WRITE(5,8) C,D
8 FORMAT(1X,'Number of rows =',I2,4X,'Number of columns =',I3)

END
```

The output for this program is shown in Figure SMG–26.

Figure SMG-26 Output Generated by SMG\$GET\_VIEWPORT\_CHAR

```
 Full Display

This is row number 1 of 4, you see
This is row number 2 of 4, you see
This is row number 3 of 4, you see
This is row number 4 of 4, you see

 Viewport

is row number 1 of 4, you see
is row number 2 of 4, you see
is row number 3 of 4, you see
is row number 4 of 4, you see

%SMG-S-WINTRUNCFIT, Viewport truncated to fit
2146623360

Row start = 1 Column start = 5
Number of rows = 4 Number of columns = 30
```

ZK-6425-GE

---

## SMG\$HOME\_CURSOR—Home Cursor

The Home Cursor routine moves the virtual cursor to the specified corner of a virtual display.

### Format

SMG\$HOME\_CURSOR display-id [,position-code]

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display in which the virtual cursor is moved. The **display-id** argument is the address of a longword that contains the display identifier. **Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### position-code

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the point to which the virtual cursor moves. The **position-code** argument is the address of a longword that contains the position code.

Valid codes for **position-code** are as follows:

| Code              | Meaning                                                                                                                                                                                                                 |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SMG\$C_UPPER_LEFT | Row 1, column 1 (the upper left-hand corner). This is the default if <b>position-code</b> is not specified.                                                                                                             |
| SMG\$C_LOWER_LEFT | Row <i>n</i> , column 1 (where <i>n</i> is the number of rows in the display). That is, the lower left-hand corner. It is useful to specify this position when accepting input for an upward-scrolling virtual display. |



| Code               | Meaning                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| SMG\$C_UPPER_RIGHT | Row 1, column $m$ (where $m$ is the number of columns in the display). That is, the upper right-hand corner.                                  |
| SMG\$C_LOWER_RIGHT | Row $n$ , column $m$ (where $n$ is the number of rows and $m$ is the number of columns in the display). That is, the lower right-hand corner. |

## Description

SMG\$HOME\_CURSOR moves the virtual cursor to a corner of the specified virtual display, according to the code specified in the **position-code** argument. You do not need to know the dimensions of the virtual display, or the virtual cursor location. If you omit the **position-code** argument, SMG\$HOME\_CURSOR moves the virtual display cursor to the upper left-hand corner of the virtual display.

## Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .   |
| SMG\$_WRONUMARG | Wrong number of arguments.    |
| SMG\$_INVARG    | Invalid argument.             |

## SMG\$INIT\_TERM\_TABLE

---

### SMG\$INIT\_TERM\_TABLE—Initialize Terminal Table

The Initialize Terminal Table routine initializes the TERMTABLE database for the terminal named, so that subsequent calls to SMG\$GET\_TERM\_DATA can extract information and command strings for that terminal.

#### Format

SMG\$INIT\_TERM\_TABLE terminal-name, termtable-address

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **terminal-name**

|               |                  |
|---------------|------------------|
| OpenVMS usage | device_name      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Specifies the name of the terminal. The **terminal-name** argument is the address of a descriptor pointing to the terminal name. The name must be an entry in TERMTABLE.EXE.

##### **termtable-address**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | address             |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Address of the entry for a particular type of terminal in TERMTABLE.EXE. The **termtable-address** argument is the address of an unsigned longword that contains the address of the terminal capabilities table.

You use this address when calling the SMG\$GET\_TERM\_DATA procedure for the specified type of terminal. **Termtable-address** is also returned by SMG\$INIT\_TERM\_TABLE\_BY\_TYPE.

#### Description

SMG\$INIT\_TERM\_TABLE initializes the TERMTABLE database for the terminal named, so that subsequent calls to SMG\$GET\_TERM\_DATA can extract information and command strings for that terminal. This routine should be used only when you perform direct (non-SMG\$) I/O to terminals.

SMG\$INIT\_TERM\_TABLE first searches for TERMTABLE.EXE in the area logically named TERM\$TABLOC. If TERMTABLE.EXE is not found there, the routine searches the global section SMG\$TERMTABLE.

**Condition Values Returned**

|                 |                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                                                    |
| SMG\$_PRISECMAP | Successful completion. The definition was found in a private TERMTABLE.                          |
| SMG\$_GBLSECMAP | Successful completion. The definition was found in the global TERMTABLE.                         |
| SMG\$_UNDTERNOP | Undefined terminal. No definition was found for the terminal and no private TERMTABLE was found. |
| SMG\$_UNDTERNOS | Undefined terminal. No definition was found for the terminal and no system TERMTABLE was found.  |
| SMG\$_UNDTERNAM | Undefined terminal name.                                                                         |

## SMG\$INIT\_TERM\_TABLE\_BY\_TYPE

---

### SMG\$INIT\_TERM\_TABLE\_BY\_TYPE—Initialize TERMTABLE by OpenVMS Terminal Type

The Initialize TERMTABLE by OpenVMS Terminal Type routine initializes the TERMTABLE database for the terminal named, so that subsequent calls to SMG\$GET\_TERM\_DATA can extract information and command strings for that terminal.

#### Format

SMG\$INIT\_TERM\_TABLE\_BY\_TYPE terminal-type ,termtable-address [,terminal-name]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### terminal-type

|               |               |
|---------------|---------------|
| OpenVMS usage | byte_signed   |
| type          | byte (signed) |
| access        | read only     |
| mechanism     | by reference  |

The device type of the terminal, as designated by a OpenVMS symbolic terminal type or by another value returned by the \$GETDVI system service. The **terminal-type** argument is the address of a signed byte that contains the terminal type.

##### termtable-address

|               |                     |
|---------------|---------------------|
| OpenVMS usage | address             |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Address of the entry for a particular type of terminal in TERMTABLE.EXE. The **termtable-address** argument is the address of an unsigned longword into which is written the address of a terminal entry.

You use this address when calling the SMG\$GET\_TERM\_DATA procedure for the specified type of terminal.

##### terminal-name

|               |                  |
|---------------|------------------|
| OpenVMS usage | device_name      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

A string into which is written the terminal name associated with the device type. The **terminal-name** argument is the address of a descriptor pointing to the string into which the terminal name is written.

## Description

SMG\$INIT\_TERM\_TABLE\_BY\_TYPE initializes the TERMTABLE database for the terminal type specified, so that subsequent calls to SMG\$GET\_TERM\_DATA can extract information and command strings for that type of terminal. This routine should be used only when you perform direct (non-SMG\$) I/O to terminals.

SMG\$INIT\_TERM\_TABLE\_BY\_TYPE first searches for TERMTABLE.EXE in the area logically named TERM\$TABLOC. If TERMTABLE.EXE is not found there, the routine searches the global section SMG\$TERMTABLE.

## Condition Values Returned

|                 |                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                                                    |
| SMG\$_PRISECMAP | Successful completion. The definition was found in a private TERMTABLE.                          |
| SMG\$_GBLSECMAP | Successful completion. The definition was found in the global TERMTABLE.                         |
| SMG\$_UNDTERNOP | Undefined terminal. No definition was found for the terminal and no private TERMTABLE was found. |
| SMG\$_UNDTERNOS | Undefined terminal. No definition was found for the terminal and no system TERMTABLE was found.  |
| SMG\$_UNDTERNAM | Undefined terminal name.                                                                         |

---

## SMG\$INSERT\_CHARS—Insert Characters

The Insert Characters routine inserts characters into a virtual display.

### Format

```
SMG$INSERT_CHARS display-id ,character-string ,start-row ,start-column [,rendition-set]
 [,rendition-complement] [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display affected. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier. **Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### character-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

The character string to be inserted. The **character-string** argument is the address of a descriptor that points to the string to be inserted.

#### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

The row position at which to begin the insertion. The **start-row** argument is the address of a signed longword that contains the row number.

#### start-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

The column position at which to begin the insertion. The **start-column** argument is the address of a signed longword that contains the column number.

**rendition-set**

OpenVMS usage mask\_longword  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                       |                                                                                                                           |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$_BLINK                           | Displays blinking characters.                                                                                             |
| SMG\$_BOLD                            | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$_REVERSE                         | Displays characters in reverse video, that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$_UNDERLINE                       | Displays underlined characters.                                                                                           |
| SMG\$_INVISIBLE                       | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$_USER1<br>through<br>SMG\$_USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

## SMG\$INSERT\_CHARS

### rendition-complement

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

### character-set

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access read only  
mechanism by reference

Specifies the character set of the string in the **character-string**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or by SMG\$CHANGE\_VIRTUAL\_DISPLAY.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

## Description

SMG\$INSERT\_CHARS inserts the specified character string at the **start-row** and **start-column** positions specified. Characters to the right of the insertion are shifted to the right. Note that any characters which do not fit on the current line are discarded. The virtual cursor remains at the character position following the last character inserted.



Original SMG routines support only ASCII(default) and DEC Special Graphics, but Chinese SMG will allow users to specify the **character-set** which enables Chinese language to be used.

In case Chinese language character set is used and the **start-column** is on the right portion of a Chinese character, the character will be changed to undefined characters. Also, if any character is discarded at the end of the line, and the discard begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

## Condition Values Returned

|                 |                                     |
|-----------------|-------------------------------------|
| SS\$_NORMAL     | Normal successful completion.       |
| SMG\$_INVARG    | Unrecognized rendition code.        |
| SMG\$_INVCOL    | Invalid column.                     |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .         |
| SMG\$_INVROW    | Invalid row.                        |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal. |
| SMG\$_WRONUMARG | Wrong number of arguments.          |
| LIB\$_INVSTRDES | Invalid string descriptor.          |

## Example

```
C+
C This DEC Fortran example program demonstrates the use of SMG$INSERT_CHARS.
C-
```

```
 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'
```

```
C+
C Use SMG$CREATE_VIRTUAL_DISPLAY to create a virtual display
C with a border.
C-
```

```
 ROWS = 7
 COLUMNS = 50
 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$_M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

```
C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
```

```
 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

```
C+
C Put data in the virtual display by calling SMG$PUT_CHARS.
C-
```

```
 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This virtual display has 7 rows and 50 columns.', 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This is a bordered virtual display.', 4, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

## SMG\$INSERT\_CHARS

```
 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$INSERT_CHARS to add a row 1 of text, starting at column 6.
C Underline these characters.
C-

 STATUS = SMG$INSERT_CHARS (DISPLAY1,
1 'This is a new row.', 1, 6, SMG$M_UNDERLINE)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

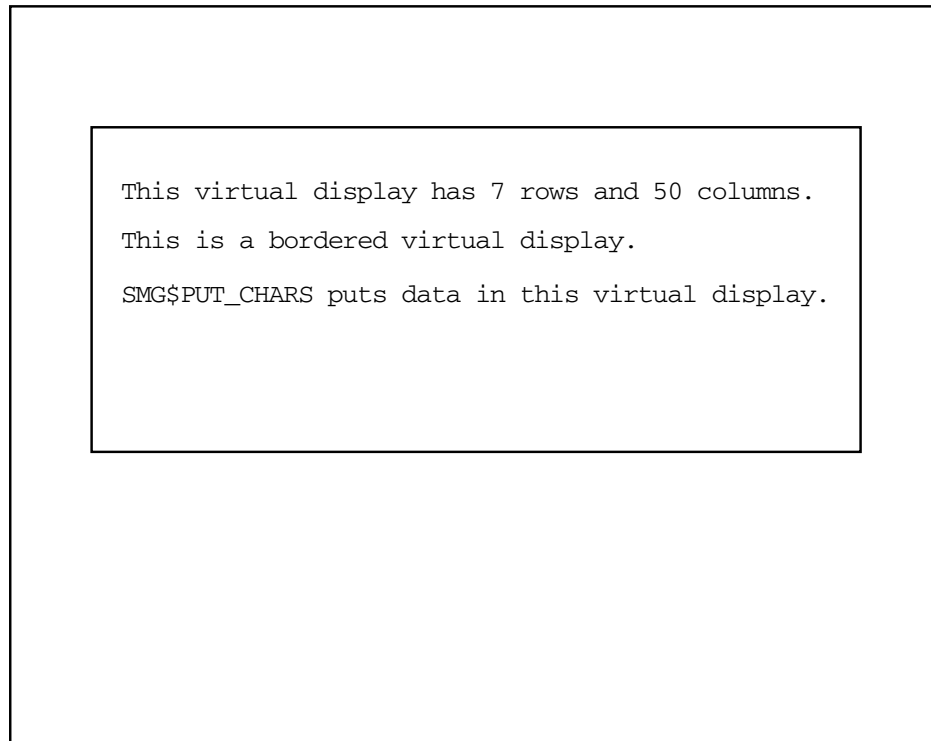
C+
C Calling SMG$INSERT_CHARS again, add text to row 6.
C Note that there will be some characters that will no
C longer fit on the line. They will be discarded. The
C new text will be bolded.
C-

 STATUS = SMG$INSERT_CHARS (DISPLAY1,
1 'to this bordered display.', 6, 28, SMG$M_BOLD)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 END
```

The output generated by this FORTRAN program before the call to SMG\$INSERT\_CHARS is shown in Figure SMG-27.

**Figure SMG-27 Output Generated by FORTRAN Program Before the Call to SMG\$INSERT\_CHARS**

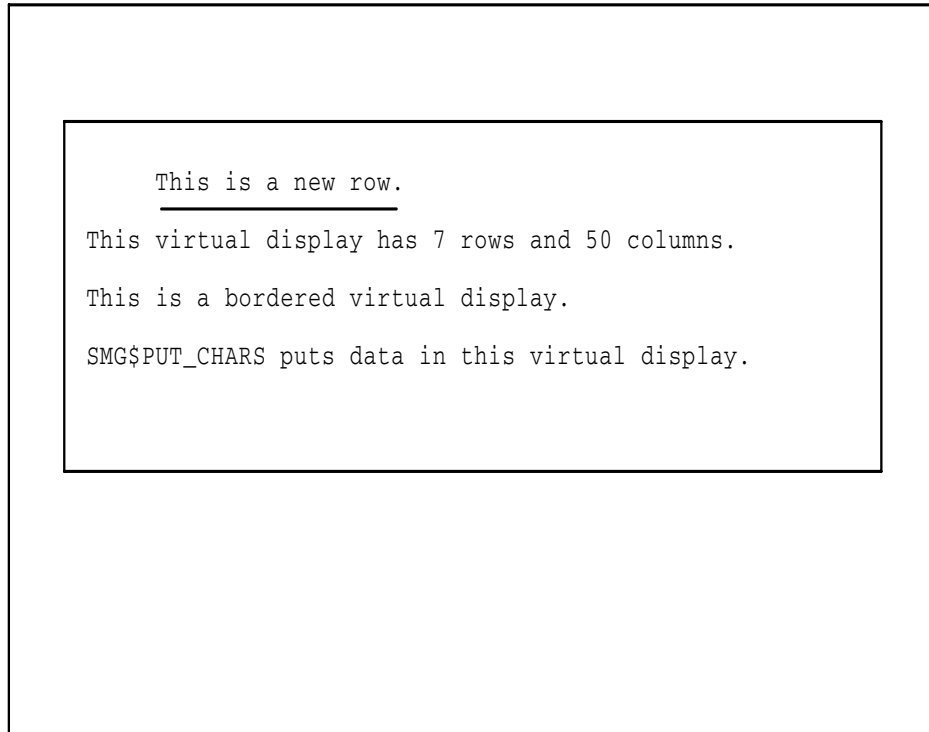


ZK-4132-GE

The output generated by this FORTRAN program after the call to SMG\$INSERT\_CHARS is shown in Figure SMG-28.

## SMG\$INSERT\_CHARS

Figure SMG-28 Output Generated by FORTRAN Program After the Call to SMG\$INSERT\_CHARS



ZK-4144-GE

---

## SMG\$INSERT\_LINE—Insert Line

The Insert Line routine inserts a line into a virtual display and scrolls the display.

### Format

```
SMG$INSERT_LINE display-id ,start-row [,character-string] [,direction] [,rendition-set]
 [,rendition-complement] [,flags] [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row number at which the string is inserted and at which scrolling begins. The **start-row** argument is the address of a signed longword that contains the row number.

## SMG\$INSERT\_LINE

### character-string

OpenVMS usage char\_string  
type character string  
access read only  
mechanism by descriptor

The character string to be inserted by SMG\$INSERT\_LINE. The **character-string** argument is the address of a descriptor pointing to this string.

### direction

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access read only  
mechanism by reference

Specifies the scrolling direction. The **direction** argument is the address of an unsigned longword that contains the direction code. Valid values are SMG\$M\_UP and SMG\$M\_DOWN. SMG\$M\_UP is the default.

### rendition-set

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                          |                                                                                                                           |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK                             | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD                              | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE                           | Displays characters in reverse video, that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE                         | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE                         | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$M_USER1<br>through SMG\$M_<br>USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

### rendition-complement

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented

with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

#### flags

OpenVMS usage mask\_longword  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Optional bit mask that specifies the action to take if the text does not fit on the line. The **flags** argument is the address of an unsigned longword that contains the flag. Valid values for **flags** are as follows:

|                  |                                         |
|------------------|-----------------------------------------|
| 0                | Does not wrap (the default)             |
| SMG\$M_WRAP_CHAR | Wraps at the last character on the line |
| SMG\$M_WRAP_WORD | Wraps at the last space on the line     |

#### character-set

OpenVMS usage longword\_unsigned  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Specifies the character set of the string in the **character-string**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or by SMG\$CHANGE\_VIRTUAL\_DISPLAY.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

## SMG\$INSERT\_LINE

### Description

SMG\$INSERT\_LINE lets you insert a line into a virtual display at a location other than the first or last line. Existing lines are scrolled in the specified direction to create an open space. If you specify a **character-string** argument, that string is written in the space created; otherwise, the new line remains blank. If the string does not span the width of the display, it is padded with blanks.

If the value of **flags** is SMG\$M\_WRAP\_WORD or SMG\$M\_WRAP\_CHAR and the specified **character-string** is longer than the width of the virtual display, SMG\$INSERT\_LINE scrolls another line and writes the excess characters in the created space. If **flags** is 0, any excess characters are discarded. The virtual cursor remains at the character position following the last character written.

See SMG\$PUT\_LINE to add lines and scroll at the first or last line in a virtual display.

In case the insertion operation specifies a Chinese language character set, and the discard begins on the right portion of Chinese character, the left portion of the character will become an undefined character.

Original SMG routines support only ASCII(default) and DEC Special Graphics, but Chinese SMG will allow users to specify the **character-set** which enables Chinese language to be used.

### Condition Values Returned

|                 |                                                              |
|-----------------|--------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                |
| SMG\$_INVARG    | Invalid argument. The specified direction is not up or down. |
| SMG\$_INVCOL    | Invalid column.                                              |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                  |
| SMG\$_INVROW    | Invalid row.                                                 |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal.                          |
| SMG\$_WRONUMARG | Wrong number of arguments.                                   |

### Example

```
C+
C This DEC Fortran example program demonstrates the use of SMG$INSERT_LINE.
C
C Include the SMG definitions. In particular, we want SMG$M_BORDER,
C SMG$M_UNDERLINE, and SMG$M_UP.
C-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF) '

C+
C Use SMG$CREATE_VIRTUAL_DISPLAY to create a virtual display
C with a border.
C-

 ROWS = 7
 COLUMNS = 50

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```



## SMG\$INSERT\_LINE

```
C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PUT_CHARS to put data in the virtual display.
C-
 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This virtual display has 7 rows and 50 columns.', 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This is a bordered virtual display.', 4, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Paste the virtual display by calling SMG$PASTE_VIRTUAL_DISPLAY.
C-
 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

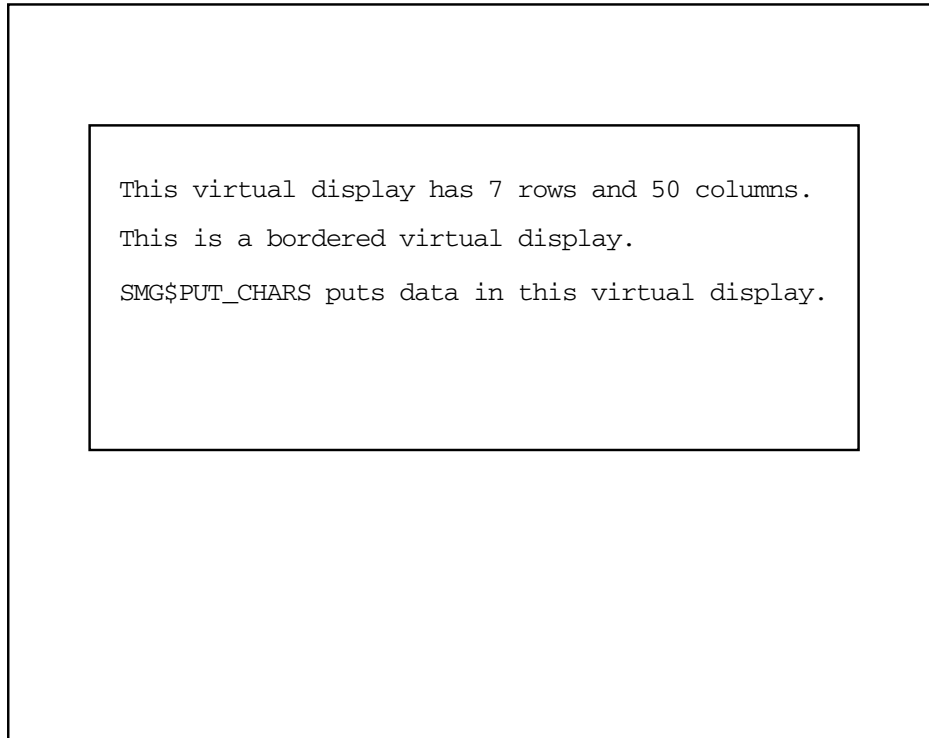
C+
C Call SMG$INSERT_LINE to add a line of text after line 6 and scroll
C the display. Also, underline the new characters.
C-
 STATUS = SMG$INSERT_LINE (DISPLAY1, 7,
1 'This is a new line.', SMGM_UP, SMGM_UNDERLINE)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 END
```

The initial output generated by this FORTRAN program is shown in Figure SMG-29.

## SMG\$INSERT\_LINE

Figure SMG–29 Output Generated Before the Call to SMG\$INSERT\_LINE



ZK-4132-GE

The output generated after the call to SMG\$INSERT\_LINE is shown in Figure SMG–30.

Figure SMG-30 Output Generated After the Call to SMG\$INSERT\_LINE

```
This virtual display has 7 rows and 50 columns.
This is a bordered virtual display.
SMG$PUT_CHARS puts data in this virtual display.
This is a new row.
```

ZK-4131-GE

## SMG\$INVALIDATE\_DISPLAY

---

### SMG\$INVALIDATE\_DISPLAY—Mark a Display as Invalid

The Mark a Display as Invalid routine marks a display as invalid and causes the entire display to be redrawn.

#### Format

SMG\$INVALIDATE\_DISPLAY display-id

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

|                   |                     |
|-------------------|---------------------|
| <b>display-id</b> |                     |
| OpenVMS usage     | identifier          |
| type              | longword (unsigned) |
| access            | read only           |
| mechanism         | by reference        |

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### Description

SMG\$INVALIDATE\_DISPLAY marks a display as invalid and redraws the entire display. You would normally use this routine after you determine that output has been written to the display without benefit of the Screen Management Facility.

This routine redraws the virtual display by invalidating the pasteboard contents of the virtual display's "footprint". If the display is occluded, only the occluded portion of the occluding virtual display is redrawn (the entire display is not redrawn).

#### Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .   |

---

## SMG\$KEYCODE\_TO\_NAME—Translate a Key Code into a Key Name

The Translate a Key Code into a Key Name routine translates the key code of a key on the keyboard into its associated key name.

### Format

SMG\$KEYCODE\_TO\_NAME key-code ,key-name

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### key-code

|               |                 |
|---------------|-----------------|
| OpenVMS usage | word_unsigned   |
| type          | word (unsigned) |
| access        | read only       |
| mechanism     | by reference    |

Specifies the key code to translate into a key name. The **key-code** argument is the address of an unsigned word that contains the key code.

**Key-code** is the same as the **word-terminator-code** argument returned by SMG\$READ\_COMPOSED\_LINE, SMG\$READ\_KEYSTROKE, and SMG\$READ\_STRING.

#### key-name

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

String containing the name of the key into which **key-code** is to be translated. The **key-name** argument is the address of a descriptor pointing to the character string containing the key name. The **key-name** argument is simply the name of the key (for example, COMMA, PERIOD, KP4, and so forth).

### Description

SMG\$KEYCODE\_TO\_NAME translates the key code of a key on the keyboard into its associated key name. This key code is the same code that is returned by the SMG\$READ\_COMPOSED\_LINE, SMG\$READ\_KEYSTROKE, and SMG\$READ\_STRING routines in the **word-terminator-code** argument. The form of **key-code** is SMG\$K\_TRM\_keyname (for example, SMG\$K\_TRM\_DELETE).

For more information on terminator values, see Table 3–1 in Chapter 3.

## SMG\$KEYCODE\_TO\_NAME

### Condition Values Returned

SS\$\_NORMAL

Normal successful completion.

SMG\$\_INVKEYNAM

Invalid **key-name**.

---

## SMG\$LABEL\_BORDER—Label a Virtual Display Border

The Label a Virtual Display Border routine supplies a label for a virtual display's border.

### Format

```
SMG$LABEL_BORDER display-id [,text] [,position-code] [,units] [,rendition-set] [,rendition-complement]
 [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### text

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

The new label for this display's border. The **text** argument is the address of a descriptor pointing to the label text. If omitted, the display is not labeled.

#### position-code

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies which of the display's borders contains the label. The **position-code** argument is the address of an unsigned longword that contains the position code.

Valid positions are as follows:

- SMG\$K\_TOP
- SMG\$K\_BOTTOM
- SMG\$K\_RIGHT
- SMG\$K\_LEFT

## SMG\$LABEL\_BORDER

SMG\$K\_RIGHT and SMG\$K\_LEFT are valid when **character-set** argument is either SMG\$C\_ASCII or SMG\$C\_SPEC\_GRAPHICS and when SMG\$C\_ASCII or SMG\$C\_SPEC\_GRAPHICS is specified in SMG\$CHANGE\_VIRTUAL\_DISPLAY.

If this argument is omitted, the label is displayed on the top border.

### units

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the character position at which the label begins within the border. The **units** argument is the address of a signed longword that contains the character position. If omitted, the label is centered in the specified border.

### rendition-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                         |                                                                                                                           |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK                            | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD                             | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE                          | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE                        | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE                        | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$M_USER1<br>through<br>SMG\$M_USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

### rendition-complement

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.



The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

#### character-set

OpenVMS usage longword\_unsigned  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Specifies the character set of the label text in **text**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or by SMG\$CHANGE\_VIRTUAL\_DISPLAY.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

## Description

SMG\$LABEL\_BORDER lets you specify text to label a virtual display. If the specified virtual display does not already have the border display attribute (SMG\$M\_BORDER), then this attribute is forced. If the label string is supplied, it replaces the current label text for this border. If you supply an empty (null) label string, the border is not labeled. If the label text (as positioned within the border) does not fit within the border, this routine returns SMG\$\_INVARG.

**Position-code** and **units** together specify the starting position of the label text within a border. If **position-code** is omitted, the default is the top border. If **units** is omitted, this routine chooses a starting position so as to center the text either horizontally or vertically, depending on the implicit or explicit position argument. If both **position-code** and **units** are omitted, the text is centered in the top border.

Note that vertical labels are available only for SMG\$C\_ASCII and SMG\$C\_SPEC\_GRAPHICS.

## SMG\$LABEL\_BORDER

Original SMG routines support only ASCII(default) and DEC Special Graphics, but Chinese SMG will allow users to specify the Chinese character set to **character-set** argument which enables Chinese language to be used.

### Condition Values Returned

|                 |                                                                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                                                                                                        |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                                                                                                          |
| SMG\$_INVARG    | Invalid argument. The combination of <b>position-code</b> , <b>units</b> , and <b>text</b> arguments resulted in a position outside the border area. |
| SMG\$_WRONUMARG | Wrong number of arguments.                                                                                                                           |

### Example

```
C+
C This DEC Fortran example program demonstrates the use of SMG$LABEL_BORDER.
C-

C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER,
C SMGK_TOP, SMGK_BOTTOM, and SMG$K_RIGHT.
C-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'

C+
C Call SMG$CREATE_VIRTUAL_DISPLAY to create virtual display number 1.
C Give it a border.
C-

 ROWS = 4
 COLUMNS = 30
 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$CREATE_VIRTUAL_DISPLAY to create virtual display number 2.
C Give it a border.
C-

 ROWS = 3
 COLUMNS = 30
 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY2, SMG$M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Create virtual display number 3. Do NOT give it a border.
C-

 ROWS = 4
 COLUMNS = 35
 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY3)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$CREATE_PASTEBOARD to create the pasteboard.
C-
```

## SMG\$LABEL\_BORDER

```
STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$PUT_CHARS to put data into the virtual displays.
C-

STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' A bordered virtual display.', 2, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS (DISPLAY2,
1 ' A bordered virtual display.', 1, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS (DISPLAY3,
1 ' Started as an unbordered display.', 2, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$LABEL_BORDER to label the virtual display borders.
C-

STATUS = SMG$LABEL_BORDER (DISPLAY1, 'Side', SMG$K_RIGHT)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$LABEL_BORDER (DISPLAY2, 'LABEL Bottom',
1 SMG$K_BOTTOM, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$LABEL_BORDER (DISPLAY3, 'Forced bordering ',
1 SMG$K_TOP)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual displays.
C-

STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 2, 10)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY2, PASTE1, 2, 45)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

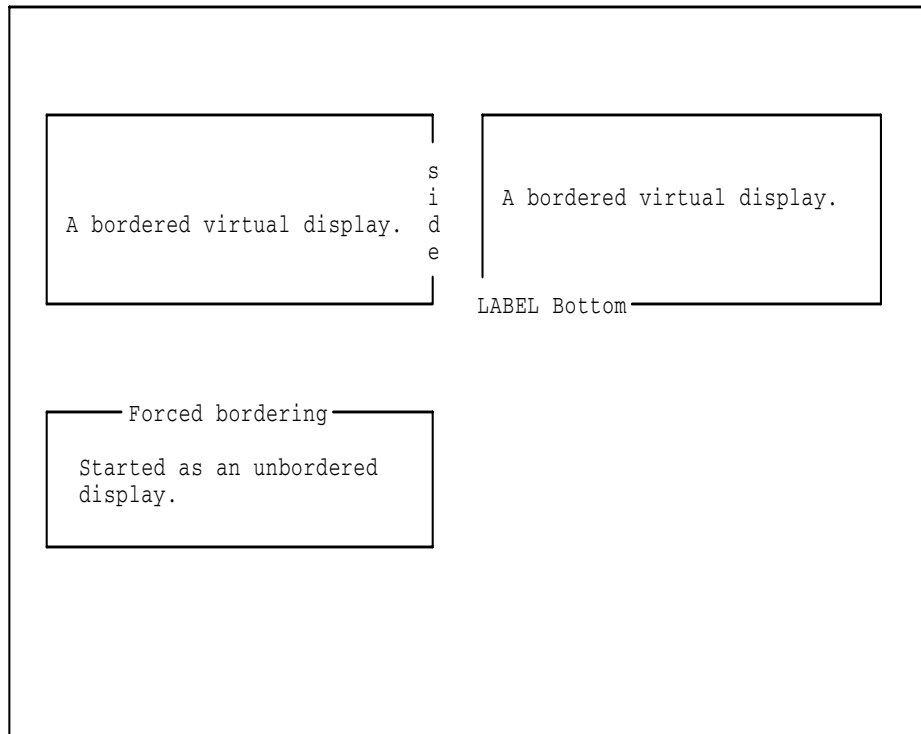
STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY3, PASTE1, 10, 5)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

END
```

The output generated by this program is shown in Figure SMG-31.

# SMG\$LABEL\_BORDER

Figure SMG-31 Output Generated by Program Calling SMG\$LABEL\_BORDER



ZK-4127-GE

---

## SMG\$LIST\_KEY\_DEFS—List Key Definitions

The List Key Definitions routine returns, one at a time, the definitions (equivalence strings) associated with specified keys in a specified key table.

### Format

```
SMG$LIST_KEY_DEFS key-table-id ,context [,key-name] [,if-state] [,attributes] [,equivalence-string]
 [,state-string]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### key-table-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the key table from which you are extracting a key definition. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

**Key-table-id** is returned by the SMG\$CREATE\_KEY\_TABLE routine.

#### context

|               |                     |
|---------------|---------------------|
| OpenVMS usage | context             |
| type          | longword (unsigned) |
| access        | modify              |
| mechanism     | by reference        |

Provides a means to extract a series of key definitions from a key table. The **context** argument is the address of an unsigned longword that contains the context variable. For the first call to this routine, you should set the **context** argument to zero.

**Context** is incremented by the SMG\$LIST\_KEY\_DEFS routine so that the next call returns the next key definition.

#### key-name

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | modify           |
| mechanism     | by descriptor    |

Identifies the key whose value you are listing. The **key-name** argument is the address of a descriptor pointing to the key name.

## SMG\$LIST\_KEY\_DEFS

### **if-state**

OpenVMS usage char\_string  
type character string  
access write only  
mechanism by descriptor

Receives the state name which qualifies the next definition in the key table. The **if-state** argument is the address of a descriptor pointing to the string into which the state name is written.

### **attributes**

OpenVMS usage mask\_longword  
type longword (unsigned)  
access write only  
mechanism by reference

Attributes of this key definition. The **attributes** argument is the address of an unsigned longword into which the key attributes are written.

Possible attributes are as follows:

|                      |                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_KEY_NOECHO    | If set, this bit specifies that <b>equiv_string</b> is not to be echoed when this key is pressed; if clear, <b>equiv_string</b> is echoed. If SMG\$M_KEY_TERMINATE is not set, SMG\$M_KEY_NOECHO is ignored.                                                                                                                                        |
| SMG\$M_KEY_TERMINATE | If set, this bit specifies that when this key is pressed (as qualified by <b>if-state</b> ), the input line is complete and more characters should not be accepted. If clear, more characters may be accepted.                                                                                                                                      |
| SMG\$M_KEY_LOCK      | If set, and if <b>state-string</b> is specified, the state name specified by <b>state-string</b> remains at the current state until explicitly changed by a subsequent keystroke whose definition includes a <b>state-string</b> . If clear, the state name specified by <b>state-string</b> remains in effect only for the next defined keystroke. |
| SMG\$M_KEY_PROTECTED | If set, this bit specifies that this key definition cannot be modified or deleted. If clear, the key definition can be modified or deleted.                                                                                                                                                                                                         |

### **equivalence-string**

OpenVMS usage char\_string  
type character string  
access write only  
mechanism by descriptor

The character string into which the equivalence string is written for the next key definition. The **equivalence-string** argument is the address of a descriptor pointing to the string into which **equivalence-string** is written.

### **state-string**

OpenVMS usage char\_string  
type character string  
access write only

mechanism by descriptor

A string into which is written the new state name, if any, set by the next key definition. The **state-string** argument is the address of a descriptor pointing to the string into which the state name is written. If this key definition sets a state, the attributes flag SMG\$M\_KEY\_SETSTATE is also set.

## Description

SMG\$LIST\_KEY\_DEFS, when called repeatedly, lets you examine all the definitions in a key table. These definitions may be used with the routine SMG\$READ\_COMPOSED\_LINE.

## Condition Values Returned

|                  |                               |
|------------------|-------------------------------|
| SS\$_NORMAL      | Normal successful completion. |
| SMG\$_INVKEYNAM  | Invalid <b>key-name</b> .     |
| SMG\$_INVKTID    | Invalid <b>key-table-id</b> . |
| SMG\$_NOMOREKEYS | No more keys in this table.   |

Any condition value returned by LIB\$COPY\_DXDX.

## SMG\$LIST\_PASTEBOARD\_ORDER

---

### SMG\$LIST\_PASTEBOARD\_ORDER—Return Pasting Information

The Return Pasting Information routine returns the Pasteboard identifier of the pasteboard to which the specified virtual display is pasted. Optionally, the pasteboard row 1 and column 1 (origins) of the virtual display are also returned.

#### Format

```
SMG$LIST_PASTEBOARD_ORDER display-id ,context ,pasteboard-id [,pasteboard-row]
 [,pasteboard-column]
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Display identifier of the virtual display that is pasted. The **display-id** argument is the address of an unsigned longword containing the identifier of this display.

##### context

|               |                     |
|---------------|---------------------|
| OpenVMS usage | context             |
| type          | longword (unsigned) |
| access        | modify              |
| mechanism     | by reference        |

Context to search. The **context** argument is the address of an unsigned longword containing this context. On the initial call, you should set **context** to zero. SMG\$LIST\_PASTEBOARD\_ORDER updates the value of **context**. The updated value should then be passed on the subsequent calls to obtain the next pasted display identifier.

##### pasteboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Receives the identifier of the pasteboard on which the virtual display is pasted. The **pasteboard-id** argument is the address of an unsigned longword containing the identifier of this pasteboard.

##### pasteboard-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |



## SMG\$LIST\_PASTEBOARD\_ORDER

Row of the pasteboard that contains row 1 of the specified virtual display. The optional **pasteboard-row** argument is the address of a signed longword containing the number of the pasteboard row that contains the first row of the virtual display.

### **pasteboard-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |

Column of the pasteboard that contains column 1 of the specified virtual display. The optional **pasteboard-column** argument is the address of a signed longword containing the number of the pasteboard column that contains the first column of the virtual display.

## Description

SMG\$LIST\_PASTEBOARD\_ORDER returns the pasteboard identifier of the pasteboard to which the specified virtual display is pasted. Optionally, the pasteboard row 1 and column 1 (origins) of the virtual displays are also returned.

SMG\$LIST\_PASTEBOARD\_ORDER returns the identifier of the first (or bottommost) pasteboard to which the specified virtual display is pasted. Call this routine in a loop, once for each succeeding pasteboard, until SMG\$\_NOPASTED is returned.

This routine is useful if, for example, you pasted a display to many pasteboards and you no longer know the pasteboard identifier of a pasteboard critical to your application.

## Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_NOPASTED  | No more displays are pasted.  |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .   |
| SMG\$_WRONUMARG | Wrong number of arguments.    |

## SMG\$LIST\_PASTING\_ORDER

---

### SMG\$LIST\_PASTING\_ORDER—Return Virtual Display Pasting Information

The Return Virtual Display Pasting Information routine returns the identifiers of the virtual displays pasted to a specified pasteboard. Optionally, the pasteboard row 1 and column 1 (origins) of the virtual displays are also returned.

#### Format

SMG\$LIST\_PASTING\_ORDER pasteboard-id ,context ,display-id [,pasteboard-row] [,pasteboard-column]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **pasteboard-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the pasteboard on which the virtual display is pasted. The **pasteboard-id** argument is the address of an unsigned longword containing the identifier of this pasteboard.

##### **context**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | context             |
| type          | longword (unsigned) |
| access        | modify              |
| mechanism     | by reference        |

Context to search. The **context** argument is the address of an unsigned longword containing this context. On the initial call, you should set **context** to zero. SMG\$LIST\_PASTING\_ORDER updates the value of **context**. The updated value should then be passed on subsequent calls to obtain the next pasted display identifier.

##### **display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Identifier of the virtual display that is pasted. The **display-id** argument is the address of an unsigned longword containing the identifier of this virtual display.

##### **pasteboard-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |

Row of the pasteboard that contains row 1 of the specified virtual display. The optional **pasteboard-row** argument is the address of a signed longword containing the number of the pasteboard row that contains the first row of the virtual display.

## pasteboard-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |

Column of the pasteboard that contains column 1 of the specified virtual display. The optional **pasteboard-column** argument is the address of a signed longword containing the number of the pasteboard column that contains the first column of the virtual display.

## Description

SMG\$LIST\_PASTING\_ORDER returns the identifiers of the virtual displays pasted to a specified pasteboard. Optionally, the pasteboard row 1 and column 1 (origins) of the virtual displays are also returned.

SMG\$LIST\_PASTING\_ORDER returns the identifier of the first (or bottommost) virtual display pasted. Call this routine in a loop, once for each succeeding pasted virtual display, until SMG\$\_NOTPASTED is returned.

This routine is useful if, for example, you created many virtual displays and you no longer know the display identifier of a virtual display critical to your application.

## Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_NOTPASTED | No more displays are pasted.   |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SMG\$_WRONUMARG | Wrong number of arguments.     |

---

## SMG\$LOAD\_KEY\_DEFS—Load Key Definitions

The Load Key Definitions routine loads a file of key definitions (DEFINE/KEY commands) into a specified key table.

### Format

```
SMG$LOAD_KEY_DEFS key-table-id ,filespec [,default-filespec] [,flags]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### **key-table-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the key table into which you are loading key definitions. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

**Key-table-id** is returned by SMG\$CREATE\_KEY\_TABLE.

#### **filespec**

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String containing the file specification for the file of DEFINE/KEY commands. The **filespec** argument is the address of a descriptor pointing to the file specification.

**default-filespec**

OpenVMS usage char\_string  
 type character string  
 access read only  
 mechanism by descriptor

String containing the default file specification for the file of DEFINE/KEY commands. The **default-filespec** argument is the address of a descriptor pointing to the default file specification. If omitted, the null string is used.

**flags**

OpenVMS usage mask\_longword  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Optional bit mask that specifies whether **filespec** is to be treated as a logical name. The **flags** argument is the address of an unsigned longword that contains the flag. If set, **flags** specifies that **filespec** should be translated, but if this is not possible, that the null string be used.

**Description**

SMG\$LOAD\_KEY\_DEFS opens and reads a file containing DEFINE/KEY commands and calls SMG\$DEFINE\_KEY for each command line in the file. Use of SMG\$LOAD\_KEY\_DEFS requires that the calling program be run under the DCL command language interpreter. This routine signals any errors encountered while processing command lines. For more information, see the SMG\$DEFINE\_KEY routine.

**Condition Values Returned**

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                         |
| SMG\$_FILTOOLON | File specification is too long (over 255 characters). |

Any condition values returned by SMG\$DEFINE\_KEY.

Any condition values returned by \$OPEN.

## SMG\$LOAD\_VIRTUAL\_DISPLAY

---

### SMG\$LOAD\_VIRTUAL\_DISPLAY—Load a Virtual Display from a File

The Load a Virtual Display from a File routine creates a new virtual display and loads it with a virtual display saved with SMG\$SAVE\_VIRTUAL\_DISPLAY.

#### Format

SMG\$LOAD\_VIRTUAL\_DISPLAY display-id [,filespec]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Identifier of the new virtual display to be loaded with the saved virtual display. The **display-id** argument is the address of an unsigned longword into which is written the new display identifier.

##### filespec

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String containing the file specification of the file in which the specified virtual display is saved. The **filespec** argument is a character string containing the file specification.

If **filespec** is omitted, SMG\$LOAD\_VIRTUAL\_DISPLAY searches for the default file specification SMGDISPLY.DAT.

#### Description

SMG\$LOAD\_VIRTUAL\_DISPLAY creates a new virtual display and loads it with a virtual display saved with SMG\$SAVE\_VIRTUAL\_DISPLAY. The new virtual display contains text, renditions, character sets, and attributes from the saved virtual display, but does not include menu, viewport, and subprocess context. The new virtual display is not pasted to any pasteboard.

## Condition Values Returned

SS\$\_NORMAL

Normal successful completion.

SMG\$\_INVDIS\_ID

Invalid **display-id**.

SMG\$\_xxxx

Any condition value returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

RMS\$\_xxxx

Any error returned by \$OPEN, \$CONNECT, \$PUT, and \$CLOSE.

## SMG\$MOVE\_TEXT

---

### SMG\$MOVE\_TEXT—Move Text from One Virtual Display to Another

The Move Text from One Virtual Display to Another routine moves a rectangle of text from one virtual display to another virtual display. Given two points in opposite corners of the rectangle, SMG\$MOVE\_TEXT determines the desired width and height of the new virtual display.

#### Format

```
SMG$MOVE_TEXT display-id ,top-left-row ,top-left-column ,bottom-right-row ,bottom-right-column
 ,display-id2 [,top-left-row2] [,top-left-column2] [,flags]
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display from which text is to be moved. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### top-left-row

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Top left row of the rectangle of text you want to move. The **top-left-row** argument is the address of an unsigned longword containing the row number.

##### top-left-column

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Top left column of the rectangle of text you want to move. The **top-left-column** argument is the address of an unsigned longword containing the column number.

##### bottom-right-row

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |



Bottom right row of the rectangle of text you want to move. The **bottom-right-row** argument is the address of an unsigned longword containing the row number.

**bottom-right-column**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Bottom right column of the rectangle of text you want to move. The **bottom-right-column** argument is the address of an unsigned longword containing the column number.

**display-id2**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display to which the text is to be moved. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

## SMG\$MOVE\_TEXT

### **top-left-row2**

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access read only  
mechanism by reference

Top left row of the rectangle that is the destination of the text you want to move. The optional **top-left-row2** argument is the address of an unsigned longword containing the row number. If you do not specify a value for **top-left-row2**, the text is moved to the current virtual cursor row.

### **top-left-column2**

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access read only  
mechanism by reference

Top left column of the rectangle that is the destination of the text you want to move. The optional **top-left-column2** argument is the address of an unsigned longword containing the column number. If you do not specify a value for **top-left-column2**, the text is moved to the current virtual cursor column.

### **flags**

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

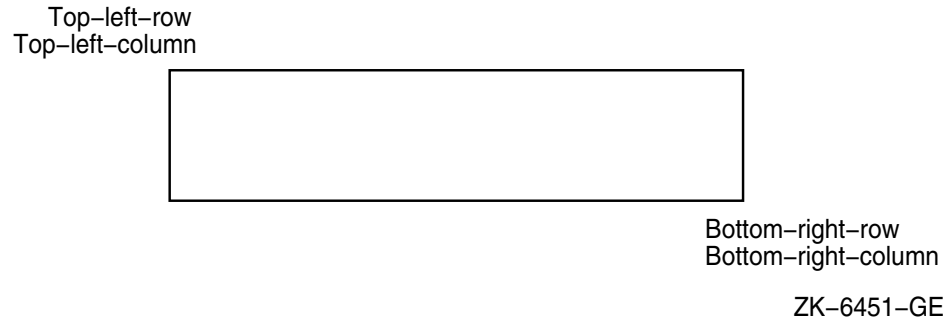
Optional bit mask that specifies the action to take when moving the specified text. The **flags** argument is the address of an unsigned longword that contains the flag. **Flags** accepts the following values:

|                  |                                        |
|------------------|----------------------------------------|
| SMG\$M_TEXT_SAVE | Does not erase the text after moving.  |
| SMG\$M_TEXT_ONLY | Moves the text but not the attributes. |

## Description

SMG\$MOVE\_TEXT moves a rectangle of text from one virtual display to another virtual display. You specify the rectangle you want to move with the **top-left-row**, **top-left-column**, **bottom-right-row**, and **bottom-right-column** arguments, as shown in Figure SMG-32. The virtual cursor positions are not changed.

**Figure SMG-32 Arguments for Moving Text**



The rectangle of text is moved to the current virtual cursor row and column of the destination virtual display. To move the text to a different position, use the optional **top-left-row2** and **top-left-column2** arguments.

By default, the attributes of the first virtual display are moved and, after the rectangle of text is moved, it is erased from the first virtual display. You can use the **flags** argument to change those default values.

In case Chinese language character set is used and the Chinese character is divided by the result of this operation, the divided parts of the Chinese character will become two undefined characters.

## Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .   |
| SMG\$_INVCOL    | Invalid column number.        |
| SMG\$_INVROW    | Invalid row number.           |
| SMG\$_WRONUMARG | Wrong number of arguments.    |

## SMG\$MOVE\_VIRTUAL\_DISPLAY

---

### SMG\$MOVE\_VIRTUAL\_DISPLAY—Move Virtual Display

The Move Virtual Display routine relocates a virtual display on a pasteboard and preserves the pasting order.

#### Format

```
SMG$MOVE_VIRTUAL_DISPLAY display-id ,pasteboard-id ,pasteboard-row ,pasteboard-column
[,top-display-id]
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display to be moved. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### **pasteboard-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard on which the movement is to take place. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

##### **pasteboard-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row of the pasteboard that is to contain row 1 of the new location of the specified virtual display. The **pasteboard-row** argument is the address of a signed longword that contains the row number.

##### **pasteboard-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column of the pasteboard that is to contain column 1 of the specified virtual display. The **pasteboard-column** argument is the address of a signed longword that contains the column number.

#### **top-display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the virtual display under which the moving **display-id** will be pasted. The **top-display-id** argument is the address of an unsigned longword containing the specified virtual display identifier. Note that the use of the **top-display-id** argument is only valid when the virtual display specified by **display-id** is not currently pasted and the virtual display specified by **top-display-id** is pasted.

## Description

SMG\$MOVE\_VIRTUAL\_DISPLAY moves a virtual display from its current position to the specified position and, if the virtual display is pasted, preserves the pasting order. If the display being moved is not currently pasted, SMG\$MOVE\_VIRTUAL\_DISPLAY presents the user with two options. By default, SMG\$MOVE\_VIRTUAL\_DISPLAY pastes the display at the top of the pasting order in the position specified.

If, however, the optional argument **top-display-id** is specified, SMG\$MOVE\_VIRTUAL\_DISPLAY pastes the virtual display being moved under the virtual display specified by **top-display-id**. In this case, the virtual display specified by **top-display-id** must already be pasted.

Note that a display cannot be moved from one pasteboard to another. However, the **pasteboard-id** is required because a given virtual display may be pasted to any number of pasteboards.

Do not use this routine if the virtual display is batched.

## Condition Values Returned

|                 |                                              |
|-----------------|----------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                  |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> .               |
| SMG\$_WRONUMARG | Wrong number of arguments.                   |
| SMG\$_ILLBATFNC | Display is being batched; illegal operation. |

## Example

Refer to the FORTRAN example shown in the SMG\$REPASTE\_VIRTUAL\_DISPLAY routine.

## SMG\$NAME\_TO\_KEYCODE

---

### SMG\$NAME\_TO\_KEYCODE—Translate a Key Name into a Key Code

The Translate a Key Name into a Key Code routine translates the key name of a key on the keyboard into its associated key code.

#### Format

SMG\$NAME\_TO\_KEYCODE key-name ,key-code

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### key-name

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String containing the name of the key to be translated into a key code. The **key-name** argument is the address of a descriptor pointing to the character string containing the key name. The **key-name** argument is simply the name of the key (for example, COMMA, PERIOD, KP4, and so forth).

##### key-code

|               |                 |
|---------------|-----------------|
| OpenVMS usage | word_unsigned   |
| type          | word (unsigned) |
| access        | write only      |
| mechanism     | by reference    |

Specifies the key code into which **key-name** is to be translated. The **key-code** argument is the address of an unsigned word that contains the key code.

**Key-code** is the same as the **word-terminator-code** argument returned by SMG\$READ\_COMPOSED\_LINE, SMG\$READ\_KEYSTROKE, and SMG\$READ\_STRING.

#### Description

SMG\$NAME\_TO\_KEYCODE translates the key name of a key on the keyboard into its associated key code. This key code is the same code that is returned by the SMG\$READ\_COMPOSED\_LINE, SMG\$READ\_KEYSTROKE, and SMG\$READ\_STRING routines in the **word-terminator-code** argument. The form of the key code returned by this routine is SMG\$K\_TRM\_keyname. For example, if you supply the key name ENTER, the key code returned by this routine is SMG\$K\_TRM\_ENTER.

For more information on terminator codes, see Table 3–1 in Chapter 3.

**Condition Values Returned**

SS\$\_NORMAL

Normal successful completion.

SMG\$\_INVKEYNAM

Invalid **key-name**.

---

## SMG\$PASTE\_VIRTUAL\_DISPLAY—Paste Virtual Display

The Paste Virtual Display routine pastes a virtual display to a pasteboard.

### Format

```
SMG$PASTE_VIRTUAL_DISPLAY display-id ,pasteboard-id [,pasteboard-row] [,pasteboard-column]
 [,top-display-id]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display to be pasted. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### pasteboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard to which the display is to be pasted. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

#### pasteboard-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row of the pasteboard that is to contain row 1 of the specified virtual display. The **pasteboard-row** argument is the address of a signed longword that contains the row number.

If **pasteboard-row** is omitted, the last row to which the virtual display was pasted (on any pasteboard) is used. If **pasteboard-row** is omitted and the virtual display was not previously pasted, then row 1 is used.

#### pasteboard-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |



mechanism           by reference

Specifies the column of the pasteboard that is to contain column 1 of the specified virtual display. The **pasteboard-column** argument is the address of a signed longword that contains the column number.

If **pasteboard-column** is omitted, the last column to which the virtual display was pasted (on any pasteboard) is used. If **pasteboard-column** is omitted and the virtual display was not previously pasted, then column 1 is used.

#### **top-display-id**

OpenVMS usage    identifier  
 type             longword (unsigned)  
 access           read only  
 mechanism        by reference

Identifier of the virtual display under which to paste **display-id**. The optional **top-display-id** argument is the address of an unsigned longword containing this identifier. Note that the virtual display specified by **top-display-id** must already be pasted.

## Description

SMG\$PASTE\_VIRTUAL\_DISPLAY places a display on a pasteboard and makes the display visible, unless the optional argument **top-display-id** is specified. If **top-display-id** is specified, SMG\$PASTE\_VIRTUAL\_DISPLAY pastes the virtual display being pasted under the virtual display specified by **top-display-id**. In this case, the virtual display specified by **top-display-id** must already be pasted.

Do not use this routine if the virtual display is batched.

## Condition Values Returned

|                 |                                              |
|-----------------|----------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                  |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> .               |
| SMG\$_WRONUMARG | Wrong number of arguments.                   |
| SMG\$_ILLBATFNC | Display is being batched; illegal operation. |

## SMG\$PASTE\_VIRTUAL\_DISPLAY

### Example

```

 0 | 1 | 2 | 3 | 4 | 5 | 6 |
1234567890123456789012345678901234567890123456789012345678901234567890

C* RPG program demonstrating SMG$PASTE_VIRTUAL_DISPLAY.
C CREPAS EXTRN' SMG$CREATE_PASTEBOARD'
C CREDIS EXTRN' SMG$CREATE_VIRTUAL_DISPLAY'
C PUTCHA EXTRN' SMG$PUT_CHARS'
C PASDIS EXTRN' SMG$PASTE_VIRTUAL_DISPLAY'
C Z-ADD0 ZERO 90
C Z-ADD1 LINCOL 90
C Z-ADD2 LINE 90
C Z-ADD5 COLUMN 90
C MOVE 'Menu' OUT 4
C* Create the pasteboard.
C CALL CREPAS
C PARM PASTID 90 WL
C PARMV ZERO
C PARM HEIGHT 90 WL
C PARM WIDTH 90 WL
C* Create the virtual display.
C CALL CREDIS
C PARM HEIGHT RL
C PARM WIDTH RL
C PARM DISPID 90 WL
C* Output the 'Menu'.
C CALL PUTCHA
C PARM DISPID RL
C PARM PARM OUT
C PARM LINE RL
C PARM COLUMN RL
C* Paste the virtual display.
C CALL PASDIS
C PARM DISPID RL
C PARM PASTID RL
C PARM LINCOL RL
C PARM LINCOL RL
C SETON LR
```

The RPG II program above displays 'Menu' beginning at line 2 column 5.

This RPG II program calls several SMG\$ routines. For another example of how to call SMG\$PASTE\_VIRTUAL\_DISPLAY, see the RPG II example in the description of SMG\$CREATE\_PASTEBOARD.

---

## SMG\$POP\_VIRTUAL\_DISPLAY—Delete a Series of Virtual Displays

The Delete a Series of Virtual Displays routine deletes a specified virtual display and all displays that were pasted on the specified pasteboard on top of the specified virtual display.

### Format

SMG\$POP\_VIRTUAL\_DISPLAY display-id ,pasteboard-id

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the lowest (first) virtual display to be deleted. The **display-id** argument is the address of an unsigned longword that contains the display identifier. All displays that are higher in the pasting order (that is, all displays that were pasted after the specified display) are deleted as well.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### pasteboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard on which the display deletions take place. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

### Description

SMG\$POP\_VIRTUAL\_DISPLAY deletes (not merely unpastes) one or more displays from the specified pasteboard, starting with the display specified and including all displays that are higher in the pasting order (that is, all displays that were pasted on top of the specified display).

## SMG\$POP\_VIRTUAL\_DISPLAY

### Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .    |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SMG\$_WRONUMARG | Wrong number of arguments.     |

---

## SMG\$PUT\_CHARS—Write Characters to a Virtual Display

The Write Characters to a Virtual Display routine writes characters in a virtual display with the text you specify.

### Format

```
SMG$PUT_CHARS display-id ,text [,start-row] [,start-column] [,flags] [,rendition-set]
 [,rendition-complement] [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### text

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Characters to be written to the virtual display. The **text** argument is the address of a descriptor pointing to the character string.

#### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row at which output begins. If **start-row** is omitted, output begins on the current row. The **start-row** argument is the address of a signed longword that contains the row number.

#### start-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

## SMG\$PUT\_CHARS

Specifies the column at which output begins. If **start-column** is omitted, output begins on the current column. The **start-column** argument is the address of a signed longword that contains the column number.

In the case that the **start-column** specifies the right portion of a Chinese character, the left portion of the character becomes an undefined character.

### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask that specifies the action to take before the specified text is output. The **flags** argument is the address of an unsigned longword bit mask that contains the flag. **Flags** accepts the following values:

|                     |                                        |
|---------------------|----------------------------------------|
| 0                   | Does not erase line (the default).     |
| SMG\$M_ERASE_TO_EOL | Erases the remaining part of the line. |
| SMG\$M_ERASE_LINE   | Erases the entire line.                |

### rendition-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                         |                                                                                                                           |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK                            | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD                             | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE                          | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE                        | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE                        | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$M_USER1<br>through<br>SMG\$M_USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

### rendition-complement

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All attributes that

can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

#### character-set

OpenVMS usage    longword\_unsigned  
 type            longword (unsigned)  
 access           read only  
 mechanism        by reference

Specifies the character set of the string in the **text**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or by SMG\$CHANGE\_VIRTUAL\_DISPLAY.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

## Description

SMG\$PUT\_CHARS writes text to the specified virtual display, possibly overwriting any existing text. Use SMG\$INSERT\_CHARS to write new text while preserving existing text.

By default, SMG\$PUT\_CHARS modifies only those character positions where new text is written. However, you can erase the line before the new text is written by specifying SMG\$M\_ERASE\_LINE for the **flags** argument, or you can erase the remainder of the line after the text is written by specifying SMG\$M\_ERASE\_TO\_EOL for the **flags** argument. The cursor remains at the character position immediately following the last text written. Note that this routine writes to a single line; excess characters at the end of the line are discarded. However, if the display has the SMG\$M\_TRUNC\_ICON attribute, SMG\$PUT\_CHAR outputs the truncation icon when the line overflows the display.

## SMG\$PUT\_CHARS

Original SMG routines support only ASCII(default) and DEC Special Graphics, but Chinese SMG will allow users to specify the **character-set** which enables Chinese language to be used.

In case Chinese language character set is used and the **start-column** is on the right portion of a Chinese character, the previous position, the left portion of the character will become undefined character. Also, if the operation terminates on the left portion of a Chinese character, the next position, the right portion of the character will be undefined character. Furthermore, if the discard begins on the right portion of a Chinese character, the left portion of the character will become undefined character.

### Condition Values Returned

|                 |                                     |
|-----------------|-------------------------------------|
| SS\$_NORMAL     | Normal successful completion.       |
| SMG\$_INVCOL    | Invalid column.                     |
| SMG\$_INVROW    | Invalid row.                        |
| SMG\$_WRONUMARG | Wrong number of arguments.          |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .         |
| LIB\$_INVSTRDES | Invalid string descriptor.          |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal. |

### Examples

```
1. C+
 C This DEC Fortran example program demonstrates the use of
 C SMG$PUT_CHARS.
 C-

 INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
 INTEGER SMG$PASTE_VIRTUAL_DISPLAY, SMG$PUT_CHARS
 INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS, BORDER, STATUS

 C+
 C Create the virtual display. To give it a border, set BORDER = 1.
 C No border would be BORDER = 0.
 C-

 ROWS = 7
 COLUMNS = 50
 BORDER = 1

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Create the pasteboard.
 C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Put data in the virtual display.
 C-

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This virtual display has 7 rows and 50 columns.', 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```



## SMG\$PUT\_CHARS

```
STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This is a bordered virtual display.', 4, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 6, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

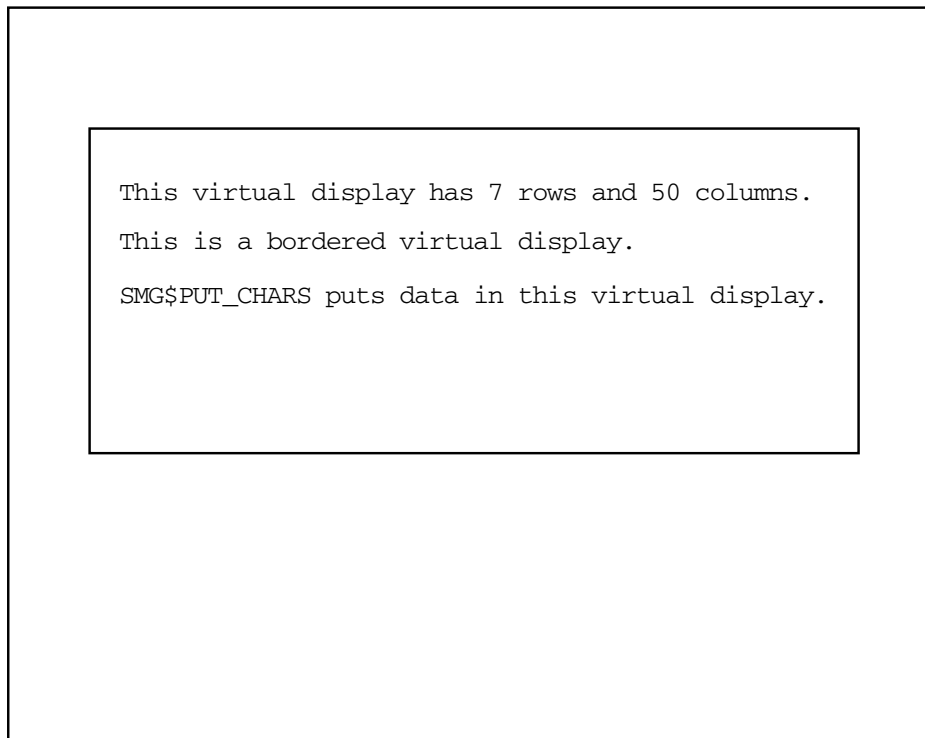
C+
C Paste the virtual display.
C-

STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

END
```

The output generated by this FORTRAN program is shown in Figure SMG-33.

**Figure SMG-33 Output of FORTRAN Program Calling SMG\$PUT\_CHARS**



ZK-4116-GE

---

## SMG\$PUT\_CHARS\_HIGHWIDE—Write Double-Height Double-Width Characters

The Write Double-Height Double-Width Characters routine writes double-height, double-width characters to a virtual display.

### Format

```
SMG$PUT_CHARS_HIGHWIDE display-id ,text [,start-row] [,start-column] [,rendition-set]
 [,rendition-complement] [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### text

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Characters to be written to the virtual display. The **text** argument is the address of a descriptor pointing to the text.

**start-row**

OpenVMS usage longword\_signed  
 type longword (signed)  
 access read only  
 mechanism by reference

Specifies the line at which output begins. The **start-row** argument is the address of a signed longword that contains the line number. If **start-row** is omitted or if it is equal to zero, output begins on the current line.

**start-column**

OpenVMS usage longword\_signed  
 type longword (signed)  
 access read only  
 mechanism by reference

Specifies the column at which output begins. The **start-column** argument is the address of a signed longword that contains the column number. If **start-column** is omitted or if it is equal to zero, output begins on the current column.

In the case that the **start-column** begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

**rendition-set**

OpenVMS usage mask\_longword  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                       |                                                                                                                           |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$_BLINK                           | Displays blinking characters.                                                                                             |
| SMG\$_BOLD                            | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$_REVERSE                         | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$_UNDERLINE                       | Displays underlined characters.                                                                                           |
| SMG\$_INVISIBLE                       | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$_USER1<br>through<br>SMG\$_USERS | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

**rendition-complement**

OpenVMS usage mask\_longword  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the

## SMG\$PUT\_CHARS\_HIGHWIDE

corresponding attribute to be complemented in the display. All attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

**character-set**

OpenVMS usage longword\_unsigned  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Specifies the character set of the text in the **text**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set which was set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or by SMG\$CHANGE\_VIRTUAL\_DISPLAY.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

**Description**

SMG\$PUT\_CHARS\_HIGHWIDE writes double-height, double-width characters to the specified virtual display. The corresponding pasteboard line cannot contain a mixture of single-height/width and double-height/width characters; if the line contains any single-height/width characters, those characters will be replaced with blanks. If the text does not span the line, it is padded with blanks. The virtual cursor remains at the character position immediately following the last text written.

Note that if the terminal does not support double-height characters, the same characters will be displayed on two successive lines.

If the display has the SMG\$M\_TRUNC\_ICON attribute, SMG\$PUT\_CHAR\_HIGHWIDE outputs the truncation icon when the line overflows the display.

In the case that the Chinese language character set is used, if the **start-column** is on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if the operation terminates on the left portion of a Chinese character, the right portion of the character will become an undefined character. Furthermore, if the discard begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

**Condition Values Returned**

|                 |                                     |
|-----------------|-------------------------------------|
| SS\$_NORMAL     | Normal successful completion.       |
| SMG\$_INVARG    | Invalid argument.                   |
| SMG\$_INVCOL    | Invalid column.                     |
| SMG\$_INVROW    | Invalid row.                        |
| LIB\$_INVSTRDES | Invalid string descriptor.          |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal. |
| SMG\$_WRONUMARG | Wrong number of arguments.          |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .         |

---

## SMG\$PUT\_CHARS\_MULTI—Put Text with Multiple Renditions to Display

The Put Text with Multiple Renditions to Display routine writes text with multiple renditions to the virtual display.

### Format

```
SMG$PUT_CHARS_MULTI display-id ,text [,start-row] [,start-column] [,flags] [,rendition-string]
 [,rendition-complement] [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the virtual display to be affected. The **display-id** argument is the address of an unsigned longword that contains this identifier.

#### text

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Text to be output. The **text** argument is the address of a descriptor pointing to the output string.

#### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Optional row number at which to start output. The **start-row** argument is the address of a signed longword containing this number. If omitted, the current row number is used.

#### start-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Optional column number at which to start output. The **start-column** argument is the address of a signed longword containing this number. If omitted, the current column number is used.

In the case that the **start-column** begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask that specifies the action to take before the specified text is output. The **flags** argument is the address of an unsigned longword that contains the flag. **Flags** accepts the following values:

|                     |                                        |
|---------------------|----------------------------------------|
| 0                   | Does not erase line (the default).     |
| SMG\$M_ERASE_TO_EOL | Erases the remaining part of the line. |
| SMG\$M_ERASE_LINE   | Erases the entire line.                |

### rendition-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Optional bit mask string that controls the video attributes. The **rendition-string** argument is the address of a descriptor pointing to the bit mask string. Each attribute set causes the corresponding attribute to be set for the corresponding byte in the text string in the display.

In the case the Chinese character set is used, if some different renditions are specified for a multi byte character, the rendition of the first byte is used for the character.

The following attributes can be specified for each byte using the **rendition-string** argument:

|                  |                                                                                                                           |
|------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK     | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD      | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE   | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |

### rendition-complement

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Optional bit mask string that controls the video attributes. The **rendition-complement** is the address of a descriptor pointing to the bit mask string. Each

## SMG\$PUT\_CHARS\_MULTI

attribute set causes the corresponding attribute to be complemented for the corresponding byte in the text string in the display.

In the case that the Chinese character set is used, if some different rendition complements are specified for a multi byte character, the rendition complement of the first byte is used for the character.

If the same bit in the same byte is specified in both the **rendition-string** and **rendition-complement** arguments, **rendition-string** is evaluated first, followed by **rendition-complement**. By using these two parameters together, you can independently control each attribute in a single routine call. On a single-attribute basis, you can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |



**character-set**

OpenVMS usage longword\_unsigned  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Specifies the character set of the string in the **text**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set which was set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or SMG\$CHANGE\_VIRTUAL\_DISPLAY. Valid values are as follows.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

**Description**

SMG\$PUT\_CHARS\_MULTI lets you write text with multiple renditions to the virtual display. No additional cursor movement sequences are added. SMG\$PUT\_CHARS\_MULTI overwrites any existing text in the positions you specify. Use SMG\$INSERT\_CHARS to write new text while preserving existing text.

By default, SMG\$PUT\_CHARS\_MULTI modifies only those character positions where new text is written. However, you can erase the line before the new text is written by specifying SMG\$M\_ERASE\_LINE for the **flags** argument, or you can erase the remainder of the line after the text is written by specifying SMG\$M\_ERASE\_TO\_EOL for the **flags** argument. The cursor remains at the character position immediately following the last text written. Note that this routine writes to a single line; excess characters at the end of the line are discarded. However, if the display has the SMG\$M\_TRUNC\_ICON attribute, SMG\$PUT\_CHARS\_MULTI outputs the truncation icon when the line overflows the display.

In the case that the Chinese language character set is used, if the **start-column** is on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if the operation terminates on the left portion of a Chinese character, the right portion of the character will become an undefined character. Furthermore, if the discard begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

**Condition Values Returned**

|                 |                                     |
|-----------------|-------------------------------------|
| SS\$_NORMAL     | Normal successful completion.       |
| SMG\$_INVARG    | Invalid argument.                   |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .         |
| SMG\$_INVCOL    | Invalid column specification.       |
| SMG\$_INVROW    | Invalid row specification.          |
| LIB\$_INVSTRDES | Invalid string descriptor.          |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal. |

## SMG\$PUT\_CHARS\_MULTI

SMG\$\_WRONUMARG

Wrong number (or combination of) arguments.

---

## SMG\$PUT\_CHARS\_WIDE—Write Double-Width Characters

The Write Double-Width Characters routine writes double-width characters to a virtual display.

### Format

```
SMG$PUT_CHARS_WIDE display-id ,text [,start-row] [,start-column] [,rendition-set]
 [,rendition-complement] [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### text

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Characters to be written to the virtual display. The **text** argument is the address of a descriptor pointing to the text.

#### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the line at which output begins. The **start-row** argument is the address of a signed longword that contains the line number. If **start-row** is omitted, output begins on the current line.

#### start-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

## SMG\$PUT\_CHARS\_WIDE

Specifies the column at which output begins. If **start-column** is omitted, output begins on the current column. The **start-column** argument is the address of a signed longword that contains the column number.

In the case that the **start-column** begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

### rendition-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                         |                                                                                                                           |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK                            | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD                             | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE                          | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE                        | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE                        | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$M_USER1<br>through<br>SMG\$M_USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

### rendition-complement

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

**character-set**

OpenVMS usage longword\_unsigned  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Specifies the character set of the text in the **text**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set which was set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or by SMG\$CHANGE\_VIRTUAL\_DISPLAY. Valid values are as follows.

| Value               | Character Set        |
|---------------------|----------------------|
| SMG\$_ASCII         | ASCII                |
| SMG\$_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$_HANYU         | DEC Hanyu            |

**Description**

SMG\$PUT\_CHARS\_WIDE writes double-width text to the specified virtual display, possibly overwriting any existing text. The line cannot contain a mixture of single- and double-width characters; if the line previously contained any single-width characters, those characters will be replaced with blanks. The virtual cursor is left at the first character position following the text written.

If the display has the SMG\$\_M\_TRUNC\_ICON attribute, SMG\$PUT\_CHAR outputs the truncation icon when the line overflows the display.

In the case that Chinese language character set is used, if the **start-column** is on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if the operation terminates on the left portion of a Chinese character, the right portion of the character will become an undefined character. Furthermore, if the discard begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

**Condition Values Returned**

|                 |                                     |
|-----------------|-------------------------------------|
| SS\$_NORMAL     | Normal successful completion.       |
| SMG\$_INVARG    | Invalid argument.                   |
| SMG\$_INVCOL    | Invalid column.                     |
| SMG\$_INVROW    | Invalid row.                        |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal. |

## SMG\$PUT\_CHARS\_WIDE

SMG\$\_WRONUMARG  
SMG\$\_INVDIS\_ID  
LIB\$\_INVSTRDES

Wrong number of arguments.  
Invalid **display-id**.  
Invalid string descriptor.

---

## SMG\$PUT\_HELP\_TEXT—Output Help Text to the Display

The Output Help Text to the Display routine retrieves and outputs the help text for the specified topic in the virtual display provided.

### Format

```
SMG$PUT_HELP_TEXT display-id [,keyboard-id] [,help-topic] [,help-library] [,rendition-set]
 [,rendition-complement] [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Display identifier of the virtual display to which the help text is written. The **display-id** argument is the address of an unsigned longword that contains this virtual display identifier. Note that this display must be pasted and cannot be occluded or batched.

#### keyboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional keyboard identifier of the virtual keyboard used for input. The **keyboard-id** argument is the address of an unsigned longword that contains this virtual keyboard identifier. If the **keyboard-id** parameter is not specified, prompting is disabled.

#### help-topic

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Optional help topic. The **help-topic** argument is the address of a descriptor pointing to the help topic string.

The help topic string pointed by **help-topic** argument must be composed by ASCII characters. If the help topic is not composed by ASCII characters, **help-topic** argument is ignored and the help facility is just invoked.

## SMG\$PUT\_HELP\_TEXT

### help-library

OpenVMS usage char\_string  
type character string  
access read only  
mechanism by descriptor

Optional help library name. The **help-library** argument is the address of a descriptor pointing to the help library name. The default is SYS\$HELP:HELPLIB.HLB.

### rendition-set

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                       |                                                                                                                           |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$_BLINK                           | Displays blinking characters.                                                                                             |
| SMG\$_BOLD                            | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$_REVERSE                         | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$_UNDERLINE                       | Displays underlined characters.                                                                                           |
| SMG\$_INVISIBLE                       | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$_USER1<br>through<br>SMG\$_USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

### rendition-complement

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the



user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| <b>Set</b> | <b>Complement</b> | <b>Action</b>                                  |
|------------|-------------------|------------------------------------------------|
| 0          | 0                 | Attribute set to default                       |
| 1          | 0                 | Attribute on                                   |
| 0          | 1                 | Attribute set to complement of default setting |
| 1          | 1                 | Attribute off                                  |

## SMG\$PUT\_HELP\_TEXT

### character-set

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access read only  
mechanism by reference

Character set specifier. The optional **character-set** argument is the address of an unsigned longword that specifies the character set to be used when writing out the help text. If this argument is omitted, the virtual display's default character set is used.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

### Description

SMG\$PUT\_HELP\_TEXT lets you retrieve and output help text for a specified topic in the virtual display specified. The text is output to the virtual display's scrolling region. If you specify the **keyboard-id** argument, you are prompted for input when the last line of the virtual scrolling region is reached. In response to this prompt you can either press the Return key to continue the display, or enter a new topic to receive help on. Note that the virtual display specified by **display-id** cannot be batched or contain a viewport. If the **keyboard-id** argument is specified, the virtual display cannot be occluded and must contain at least three rows.

The help topic string in other than ASCII character set is not supported.

### Condition Values Returned

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                                    |
| SMG\$_NOTPASTED | The virtual display specified by <b>display-id</b> is not pasted.                |
| SMG\$_ILLBATFNC | The virtual display or pasteboard was batched.                                   |
| SMG\$_INVARG    | Invalid argument.                                                                |
| SMG\$_INVDIS_ID | The <b>display-id</b> is illegal, has an associated viewport, or is occluded.    |
| LIB\$_xxxx      | Any condition value returned by LIB\$FIND_IMAGE_SYMBOL.                          |
| LBR\$_xxxx      | Any condition value returned by LBR\$OUTPUT_HELP.                                |
| SMG\$_xxxx      | Any condition value returned by SMG\$SET_CURSOR_ABS or SMG\$CHECK_FOR_OCCLUSION. |

---

## SMG\$PUT\_LINE—Write Line to Virtual Display

The Write Line to Virtual Display routine writes a line of text to a virtual display, beginning at the current virtual cursor position.

### Format

```
SMG$PUT_LINE display-id ,text [,line-advance] [,rendition-set] [,rendition-complement] [,flags]
 [,character-set] [,direction]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### text

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

The characters to be written to the virtual display. The **text** argument is the address of a descriptor pointing to the text.

#### line-advance

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the number of lines to advance after output. The **line-advance** argument is the address of a signed longword that contains the number of lines to advance. The default is 1. If you specify 0 for **line-advance**, SMG\$PUT\_LINE overwrites any existing text.

#### rendition-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

## SMG\$PUT\_LINE

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                         |                                                                                                                           |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK                            | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD                             | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE                          | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE                        | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE                        | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$M_USER1<br>through<br>SMG\$M_USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

### rendition-complement

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask that specifies the action to be taken if the text does not fit on the line. The **flags** argument is the address of an unsigned longword that contains the flag. The **flags** argument accepts the following values:

|                  |                                          |
|------------------|------------------------------------------|
| 0                | Does not wrap (the default).             |
| SMG\$M_WRAP_CHAR | Wraps at the last character on the line. |
| SMG\$M_WRAP_WORD | Wraps at the last space on the line.     |

#### character-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the character set of the text in the **text**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or by SMG\$CHANGE\_VIRTUAL\_DISPLAY.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

#### direction

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the direction to scroll, if scrolling is necessary. The **direction** argument is the address of an unsigned longword that contains the direction code. Valid values are SMG\$M\_UP and SMG\$M\_DOWN. SMG\$M\_UP is the default.

## Description

SMG\$PUT\_LINE writes lines of text to the virtual display, beginning at the current line. Once text reaches the bottom or top line (depending on the scrolling direction), subsequent calls to this routine cause the display to scroll. SMG\$PUT\_LINE writes out the entire line, starting at the current virtual cursor position. If the caller's text does not span the entire line, the line is padded with blanks.

If **flags** specifies wrapping, lines are scrolled **line-advance** times to make room for the overflow characters in the "next" line. The "next" line is determined by the scrolling **direction**. If **flags** does not specify wrapping, excess characters are discarded.

Following a call to SMG\$PUT\_LINE, the virtual cursor position is set to column 1 of the next line where output should occur. The next line where output should occur is determined by the **line-advance** and **direction** arguments; **line-advance** defaults to 1 so that subsequent calls to SMG\$PUT\_LINE do not cause overprinting.

Other SMG\$ procedures that can be used to write lines of text to a virtual display are SMG\$PUT\_LINE\_WIDE, SMG\$PUT\_LINE\_HIGHWIDE, SMG\$PUT\_LINE\_MULTI, and SMG\$INSERT\_LINE.

## SMG\$PUT\_LINE

Original SMG routines support only ASCII(default) and DEC Special Graphics, but Chinese SMG will allow users to specify the **character-set** which enables Chinese language to be used.

In case Chinese language character set is used, and the current virtual cursor position is on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if the discard begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

### Condition Values Returned

|                 |                                     |
|-----------------|-------------------------------------|
| SS\$_NORMAL     | Normal successful completion.       |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .         |
| SMG\$_WRONUMARG | Wrong number of arguments.          |
| LIB\$_INVSTRDES | Invalid string descriptor.          |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal. |

### Examples

```
1. C+
 C This DEC Fortran example program demonstrates the use of SMG$PUT_LINE.
 C-

 C+
 C Include the SMG definitions. In particular, we want SMG$_M_BORDER and
 C SMG$_M_UNDERLINE.
 C-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'
 CHARACTER*30 TEXT(3)

 C+
 C Create a virtual display with a border.
 C-

 ROWS = 7
 COLUMNS = 50

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$_M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Create the pasteboard.
 C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Put data in the virtual display.
 C-

 TEXT(1) = 'This virtual display has 7'
 TEXT(2) = 'rows and 50 columns.'
 TEXT(3) = 'Text entered by SMG$PUT_LINE.'

 C+
 C After the first line of text is printed, call SMG$PUT_LINE to
 C advance two lines.
 C-
```

```
STATUS = SMG$PUT_LINE (DISPLAY1, TEXT(1), 2)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Now, use SMG$PUT_LINE to underline the next line of text.
C Notice that 30 characters are being underlined. Advance 1
C line of text after displaying the line.
C-

STATUS = SMG$PUT_LINE (DISPLAY1, TEXT(2), 1, SMG$M_UNDERLINE)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Display the third line of text.
C-

STATUS = SMG$PUT_LINE (DISPLAY1, TEXT(3))
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Paste the virtual display.
C-

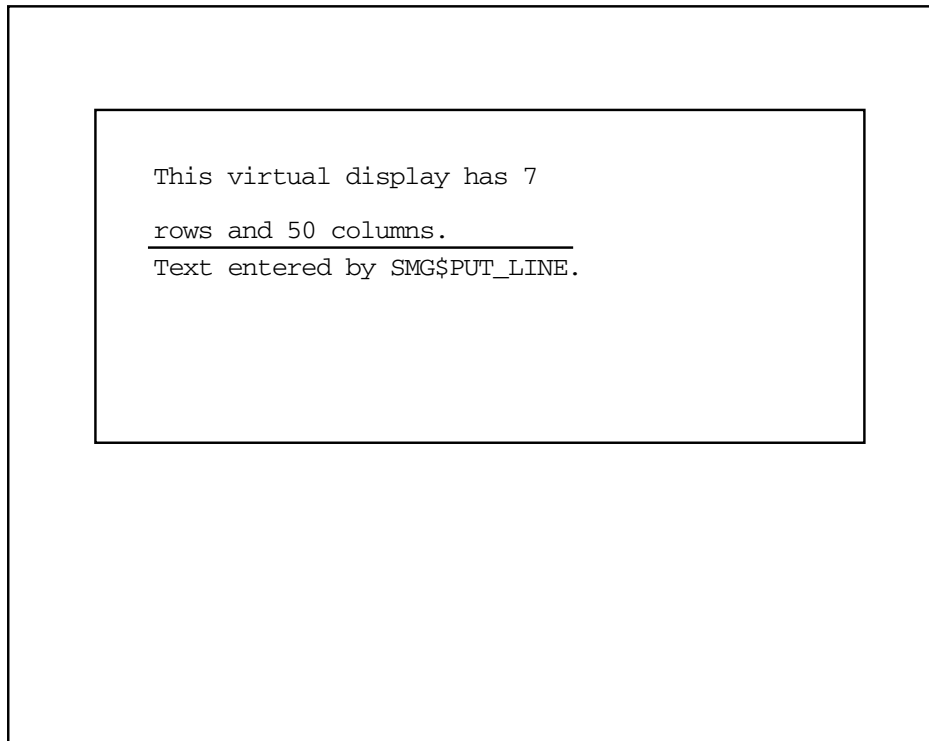
STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

END
```

The output generated by this FORTRAN program is shown in Figure SMG-34.

## SMG\$PUT\_LINE

Figure SMG-34 Output Generated by FORTRAN Program Calling SMG\$PUT\_LINE



ZK-4135-GE

The following program shows the use of the new **direction** argument to SMG\$PUT\_LINE. This new capability has made the routine SMG\$PUT\_WITH\_SCROLL obsolete.

```
2. C+
C This DEC Fortran example program demonstrates the use of the DIRECTION
C parameter in the SMG$PUT_LINE routine.
C
C The DIRECTION parameter in SMG$PUT_LINE makes SMG$PUT_WITH_SCROLL
C an obsolete routine. This example is the same as the SMG$PUT_WITH_SCROLL
C routine, except that the calls to SMG$PUT_WITH_SCROLL have been
C replaced by calls to SMG$PUT_LINE.
C-
 INCLUDE '$SMGDEF'
 IMPLICIT INTEGER*4 (A-Z)

C+
C Call SMG$CREATE_PASTEBOARD to establish the terminal screen
C as a pasteboard.
C-
 STATUS = SMG$CREATE_PASTEBOARD (NEW_PID)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))

C+
C Using SMG$CREATE_VIRTUAL_DISPLAY, establish a virtual display region.
C-
 STATUS = SMG$CREATE_VIRTUAL_DISPLAY (5,80,DISPLAY_ID)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
```



## SMG\$PUT\_LINE

```
C+
C Paste the virtual display to the screen, starting at
C row 10, column 15, by calling SMG$PASTE_VIRTUAL_DISPLAY.
C-
 STATUS = SMG$PASTE_VIRTUAL_DISPLAY(DISPLAY_ID,NEW_PID,10,15)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Define a scrolling region through a call to
C SMG$SET_DISPLAY_SCROLL_REGION.
C-
 STATUS = SMG$SET_DISPLAY_SCROLL_REGION(DISPLAY_ID,1,5)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
C+
C Call SMG$PUT_LINE and SMG$ERASE_LINE to write three
C scrolling lines to the screen. The first line will be underlined,
C the second blinking, and the third in reverse video.
C-
 DO I = 1,10
 IF ((I/2) + (I/2) .EQ. I) THEN
 DIR = SMG$M_UP
 ELSE
 DIR = SMG$M_DOWN
 ENDF
 STATUS = SMG$PUT_LINE (DISPLAY_ID,
1 'This line is underlined',,SMG$M_UNDERLINE,,,DIR)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
 STATUS = SMG$ERASE_LINE(DISPLAY_ID)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
 STATUS = SMG$PUT_LINE (DISPLAY_ID,'This line is blinking', ,
1 SMG$M_BLINK,,,DIR)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
 STATUS = SMG$ERASE_LINE (DISPLAY_ID)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
 STATUS = SMG$PUT_LINE (DISPLAY_ID,'This line is reverse
1 video',,SMG$M_REVERSE,,,DIR)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
 STATUS = SMG$ERASE_LINE (DISPLAY_ID)
 IF (.NOT. STATUS) CALL LIB$STOP(%VAL(STATUS))
 ENDDO
 END
```

---

## SMG\$PUT\_LINE\_HIGHWIDE—Write Double-Height and Double-Width Line

The Write Double-Height and Double-Width Line routine writes a line of text with double-height and double-width characters.

### Format

```
SMG$PUT_LINE_HIGHWIDE display-id ,text [,line-advance] [,rendition-set] [,rendition-complement]
 [,flags] [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Display identifier. The **display-id** argument is the address of an unsigned longword that contains the display identifier of the virtual display.

#### text

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Text to be output. The **text** argument is the address of the descriptor pointing to the output string.

#### line-advance

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Number of lines to advance. The **line-advance** argument is the address of a signed longword that contains the number of lines to advance after the output. This argument is optional.

#### rendition-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute

to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                         |                                                                                                                           |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK                            | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD                             | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE                          | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE                        | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE                        | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$M_USER1<br>through<br>SMG\$M_USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

**rendition-complement**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

**flags**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

## SMG\$PUT\_LINE\_HIGHWIDE

Optional bit mask that specifies the action to be taken if the text does not fit on the line. The **flags** argument is the address of an unsigned longword that contains the flag. **Flags** accepts the following values:

|                  |                                          |
|------------------|------------------------------------------|
| 0                | Does not wrap (the default).             |
| SMG\$M_WRAP_CHAR | Wraps at the last character on the line. |
| SMG\$M_WRAP_WORD | Wraps at the last space on the line.     |

### character-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the character set of the text in the **text**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set which was set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or SMG\$CHANGE\_VIRTUAL\_DISPLAY. Valid values are as follows.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

## Description

SMG\$PUT\_LINE\_HIGHWIDE is used to write lines of text with double-height and double-width characters to the virtual display. SMG\$PUT\_LINE\_HIGHWIDE writes from the current virtual cursor position to the end of the line. If your text does not span to the end of the line, blank spaces are added.

Treatment of text that exceeds the rightmost bounds of the display depends on the **flags** argument. If **flags** specifies wrapping, lines are scrolled **line-advance** times to make room for the overflow characters in the “next” line. If wrapping is not specified, overflow characters are lost.

Following a call to SMG\$PUT\_LINE\_HIGHWIDE, the virtual cursor position is set to column 1 of the next line where output should occur. The next line where output should occur is determined by the **line-advance** argument. **Line-advance** defaults to 2 so that subsequent calls to SMG\$PUT\_LINE\_HIGHWIDE do not cause overprinting.

In the case that the Chinese language character set is used, if the current virtual cursor position is on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if the discard begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

## Condition Values Returned

|                 |                                             |
|-----------------|---------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.               |
| SMG\$_INVARG    | Invalid argument.                           |
| SMG\$_WRONUMARG | Wrong number (or combination of) arguments. |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal.         |
| LIB\$_INVSTRDES | Invalid string descriptor.                  |

---

## SMG\$PUT\_LINE\_MULTI—Write Line with Multiple Renditions to Display

The Write Line with Multiple Renditions to Display routine writes lines with multiple renditions to the virtual display, optionally followed by cursor movement sequences.

### Format

```
SMG$PUT_LINE_MULTI display-id ,text ,rendition-string [,rendition-complement] [,line-advance] [,flags]
 [,direction] [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the virtual display to be affected. The **display-id** argument is the address of an unsigned longword that contains this identifier.

#### text

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Text to be output. The **text** argument is the address of a descriptor pointing to the output string.

#### rendition-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Optional bit mask string that controls the video attributes. The **rendition-string** argument is the address of a descriptor pointing to the bit mask string. Each attribute set causes the corresponding attribute to be set for the corresponding byte in the text string in the display.

In the case that the Chinese character set is used, if some different renditions are specified for a multi byte character, the rendition of the first byte is used for the character.

The following attributes can be specified for each byte using the **rendition-string** argument:

|                  |                                                                                                                           |
|------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK     | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD      | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE   | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |

#### rendition-complement

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Optional bit mask string that controls the video attributes. The **rendition-complement** is the address of a descriptor pointing to the bit mask string. Each attribute set causes the corresponding attribute to be complemented for the corresponding byte in the text string in the display.

In the case that the Chinese character set is used, if some different rendition complements are specified for a multi byte character, the rendition complement of the first byte is used for the character.

If the same bit in the same byte is specified in both the **rendition-string** and **rendition-complement** arguments, **rendition-string** is evaluated first, followed by **rendition-complement**. By using these two parameters together, you can independently control each attribute in a single routine call. On a single-attribute basis, you can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

#### line-advance

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Optional number of lines to advance after output. The **line-advance** argument is the address of a signed longword containing this number.

#### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

## SMG\$PUT\_LINE\_MULTI

Optional bit mask that specifies the action to take if the text does not fit on the line. The **flags** argument is the address of an unsigned longword that contains the flag. **Flags** accepts the following values:

|                  |                                          |
|------------------|------------------------------------------|
| 0                | Does not wrap (the default).             |
| SMG\$M_WRAP_CHAR | Wraps at the last character on the line. |
| SMG\$M_WRAP_WORD | Wraps at the last space on the line.     |

### direction

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional direction specifier. The **direction** argument is the address of an unsigned longword that contains the direction code specifying the scrolling direction, if scrolling is necessary. Valid values for **direction** are as follows:

- SMG\$M\_UP
- SMG\$M\_DOWN

SMG\$M\_UP is the default.

### character-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the character set of the text in the **text**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set which was set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or SMG\$CHANGE\_VIRTUAL\_DISPLAY. Valid values are as follows.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

## Description

SMG\$PUT\_LINE\_MULTI lets you write lines to the virtual display with multiple renditions, optionally followed by cursor movement sequences. SMG\$PUT\_LINE\_MULTI writes from the current virtual cursor position to the end of the line. If the text does not span to the end of the line, the remaining portion of the line is filled with blanks.

The treatment of text that extends beyond the rightmost bounds of the virtual display depends on the value of the **flags** argument. If **flags** specifies wrapping, lines are scrolled **line-advance** times to make room for the overflow characters in the “next” line. The “next” line is determined by the scrolling **direction**. If **flags** does not specify wrapping, excess characters are discarded.



Following a call to SMG\$PUT\_LINE\_MULTI, the virtual cursor position is set to column 1 of the next line where output should occur. The next line where output should occur is determined by the **line-advance** and **direction** arguments; **line-advance** defaults to 1 so that subsequent calls to SMG\$PUT\_LINE\_MULTI do not cause overprinting.

In the case that the Chinese language character set is used, if the current virtual cursor position is on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if the discard begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

## Condition Values Returned

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                           |
| SMG\$_WRONUMARG | Wrong number (or combination of) arguments.             |
| SMG\$_INVARG    | A negative value for <b>line-advance</b> was specified. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                             |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal.                     |
| LIB\$_INVSTRDES | Invalid string descriptor.                              |

## Example

```

10
!+
!This VAX BASIC example demonstrates the capabilities of the
!SMG$PUT_LINE_MULTI routine.
!-

OPTION TYPE = EXPLICIT

EXTERNAL SUB LIB$STOP (LONG BY VALUE)
EXTERNAL LONG FUNCTION SMG$CREATE_PASTEBOARD (LONG)
EXTERNAL LONG FUNCTION SMG$CREATE_VIRTUAL_DISPLAY (LONG, LONG, &
LONG, LONG)
EXTERNAL LONG FUNCTION SMG$PASTE_VIRTUAL_DISPLAY (LONG, LONG, &
LONG, LONG)
EXTERNAL LONG FUNCTION SMG$PUT_LINE (LONG, STRING)
EXTERNAL LONG FUNCTION SMG$PUT_LINE_MULTI (LONG, STRING, STRING, &
STRING, LONG, LONG)

DECLARE LONG pasteboard_id, display_id, display2_id, &
index, ret_status

MAP (rend) STRING dummy = 32
MAP DYNAMIC (rend) BYTE i_rend(15), STRING rendition
REMAP (rend) i_rend(), rendition

EXTERNAL BYTE CONSTANT SMG$M_BOLD
EXTERNAL BYTE CONSTANT SMG$M_REVERSE
EXTERNAL BYTE CONSTANT SMG$M_BLINK
EXTERNAL BYTE CONSTANT SMG$M_UNDERLINE
EXTERNAL LONG CONSTANT SMG$M_BORDER

FOR index = 0 TO 3
i_rend(index) = SMG$M_REVERSE
NEXT index

FOR index = 4 TO 7
i_rend(index) = SMG$M_BOLD
NEXT index

```

## SMG\$PUT\_LINE\_MULTI

```
FOR index = 8 to 11
 i_rend(index) = SMG$M_UNDERLINE
NEXT index

FOR index = 12 TO 15
 i_rend(index) = SMG$M_BLINK
NEXT index

REMAP (rend) rendition = 16, i_rend()

ret_status = SMG$CREATE_PASTEBOARD (pasteboard_id)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$CREATE_VIRTUAL_DISPLAY (4,10,display_id, &
 SMG$M_BORDER BY REF)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$PASTE_VIRTUAL_DISPLAY (display_id, pasteboard_id, &
 2 BY REF, 30 BY REF)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

ret_status = SMG$PUT_LINE_MULTI (display_id, '1234567890123456', &
 rendition,,,1)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP(ret_status BY VALUE)
END IF

END
```

This example shows the use of SMG\$PUT\_LINE\_MULTI. In the first line of output, the characters “1234” appear in reverse video. The characters “5678” are highlighted, and the characters “90” are underlined. In the second line of output, the characters “12” are underlined and the characters “3456” are blinking.

---

## SMG\$PUT\_LINE\_WIDE—Write Double-Width Line

The Write Double-Width Line routine writes a line of double-width text to a virtual display.

### Format

```
SMG$PUT_LINE_WIDE display-id ,text [,line-advance] [,rendition-set] [,rendition-complement] [,flags]
 [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display affected. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### text

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Characters to be written to the virtual display. The **text** argument is the address of a descriptor pointing to the text.

#### line-advance

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the number of lines to advance after output. The **line-advance** argument is the address of a signed longword integer that contains the number of lines to advance.

#### rendition-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute

## SMG\$PUT\_LINE\_WIDE

to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                         |                                                                                                                           |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK                            | Displays blinking characters.                                                                                             |
| SMG\$M_BOLD                             | Displays characters in higher-than-normal intensity.                                                                      |
| SMG\$M_REVERSE                          | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.        |
| SMG\$M_UNDERLINE                        | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE                        | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$M_USER1<br>through<br>SMG\$M_USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

### rendition-complement

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask that specifies the action to take if the text does not fit on the line. The **flags** argument is the address of an unsigned longword that contains the flag. **Flags** accepts the following values:

|                  |                                          |
|------------------|------------------------------------------|
| 0                | Does not wrap (the default).             |
| SMG\$M_WRAP_CHAR | Wraps at the last character on the line. |
| SMG\$M_WRAP_WORD | Wraps at the last space on the line.     |

#### character-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the character set of the text in the **text**. The **character-set** argument is the address of an unsigned longword that contains the character set code. If this argument is omitted, the default character set will be the character set set up by SMG\$CREATE\_VIRTUAL\_DISPLAY or by SMG\$CHANGE\_VIRTUAL\_DISPLAY. Valid values are as follows.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

## Description

SMG\$PUT\_LINE\_WIDE writes lines of double-width text to the virtual display. SMG\$PUT\_LINE\_WIDE writes out the entire line, starting at the current virtual cursor position. If the caller's text does not span the entire line, the line is filled with blanks.

If the **flags** argument specifies wrapping, lines are scrolled **line-advance** times to make room for the overflow characters in the "next" line. If **flags** does not specify wrapping, excess characters are discarded.

Following a call to SMG\$PUT\_LINE\_WIDE, the virtual cursor position is set to column 1 of the next line where output should occur. The next line where output should occur is determined by the **line-advance** argument; **line-advance** defaults to 1 so that subsequent calls to SMG\$PUT\_LINE\_WIDE will not cause overprinting.

Other routines that you can use to write text to a virtual display are SMG\$PUT\_LINE and SMG\$PUT\_LINE\_HIGHWIDE.

In the case that the Chinese language character set is used, if the current virtual cursor position is on the right portion of a Chinese character, the left portion of the character will become an undefined character. Also, if the discard begins on the right portion of a Chinese character, the left portion of the character will become an undefined character.

## SMG\$PUT\_LINE\_WIDE

### Condition Values Returned

|                 |                                     |
|-----------------|-------------------------------------|
| SS\$_NORMAL     | Normal successful completion.       |
| SMG\$_INVARG    | Invalid argument.                   |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .         |
| SMG\$_WRONUMARG | Wrong number of arguments.          |
| SMG\$_WILUSERMS | Pasteboard is not a video terminal. |
| LIB\$_INVSTRDES | Invalid string descriptor.          |

### Example

```
C+
C This DEC Fortran example program demonstrates the use of
C SMG$PUT_LINE_WIDE.
C
C Include the SMG definitions. In particular, we want SMG$M_BORDER and
C SMG$M_UNDERLINE.
C-

 INCLUDE '($SMGDEF)'
 INTEGER SMG$CREATE_VIRTUAL_DISPLAY, SMG$CREATE_PASTEBOARD
 INTEGER SMG$PASTE_VIRTUAL_DISPLAY, SMG$PUT_LINE_WIDE
 INTEGER DISPLAY1, PASTE1, ROWS, COLUMNS, STATUS
 CHARACTER*34 TEXT(3)

C+
C Create a virtual display with a border by calling
C SMG$CREATE_VIRTUAL_DISPLAY.
C-

 ROWS = 7
 COLUMNS = 70

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$PUT_LINE to put data in the virtual display.
C-

 TEXT(1) = 'This virtual display has 7'
 TEXT(2) = 'rows and 70 columns.'
 TEXT(3) = 'Text entered by SMG$PUT_LINE_WIDE.'

C+
C After the first line of text is printed, advance two lines.
C-

 STATUS = SMG$PUT_LINE_WIDE (DISPLAY1, TEXT(1), 2)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Underline the next line of text. Notice that 34 characters are being
C underlined. Advance 1 line of text after displaying the line.
C-
```

## SMG\$PUT\_LINE\_WIDE

```
 STATUS = SMG$PUT_LINE_WIDE (DISPLAY1, TEXT(2), 1,
1 SMG$M_UNDERLINE)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Display the third line of text.
C-

 STATUS = SMG$PUT_LINE_WIDE (DISPLAY1, TEXT(3))
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

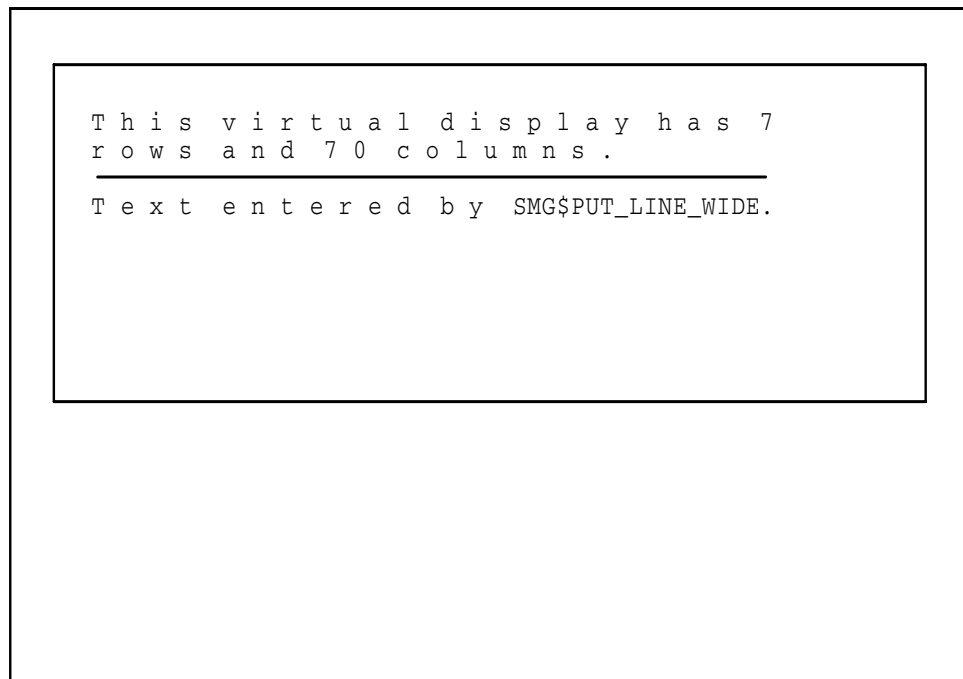
C+
C Paste the virtual display using SMG$PASTE_VIRTUAL_DISPLAY.
C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 5)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 END
```

The output generated by this FORTRAN program is shown in Figure SMG-35.

**Figure SMG-35 Output Generated by FORTRAN Program Calling SMG\$PUT\_LINE\_WIDE**



```
 T h i s v i r t u a l d i s p l a y h a s 7
 r o w s a n d 7 0 c o l u m n s .

 T e x t e n t e r e d b y S M G $ P U T _ L I N E _ W I D E .
```

ZK-4143-GE

## SMG\$PUT\_STATUS\_LINE

---

### SMG\$PUT\_STATUS\_LINE—Output Line of Text to Hardware Status Line

The Output Line of Text to Hardware Status Line routine outputs a line of text to the hardware status line.

#### Format

SMG\$PUT\_STATUS\_LINE pasteboard-id ,text [,character-set]

#### Returns

OpenVMS usage cond\_value  
type longword (unsigned)  
access write only  
mechanism by value

#### Arguments

##### pasteboard-id

OpenVMS usage identifier  
type longword (unsigned)  
access read only  
mechanism by reference

Specifies the pasteboard containing the hardware status line. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

##### text

OpenVMS usage char\_string  
type character string  
access read only  
mechanism by descriptor

The characters to be written to the hardware status line. The **text** argument is the address of a descriptor pointing to the text.

##### character-set

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access read only  
mechanism by reference

Character set specifier. The optional **character-set** argument is the address of an unsigned longword that specifies the character set to be used when writing out the text to the hardware status line. Valid values are as follows.

---

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

---



## Description

The SMG\$PUT\_STATUS\_LINE routine outputs a line of text to the terminal's hardware status line. Some terminals have a hardware status line at the bottom (25th line) of the screen. If this status line has been set as "host writable", you can use this routine to output a line of text to the status line. (If the hardware status line is not available, the error SMG\$\_OPNOTSUP is returned.) The text is output in reverse video.

## Condition Values Returned

|                 |                                    |
|-----------------|------------------------------------|
| SS\$_NORMAL     | Normal successful completion.      |
| SMG\$_WRONUMARG | Wrong number of arguments.         |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> .     |
| LIB\$_INVARG    | Invalid argument.                  |
| SMG\$_OPNOTSUP  | No hardware status line available. |

## SMG\$READ\_COMPOSED\_LINE

---

### SMG\$READ\_COMPOSED\_LINE—Read Composed Line

The Read Composed Line routine reads a line of input composed of normal keystrokes and equivalence strings.

#### Format

```
SMG$READ_COMPOSED_LINE keyboard-id [,key-table-id] ,resultant-string [,prompt-string]
 [,resultant-length] [,display-id] [,flags] [,initial-string] [,timeout]
 [,rendition-set] [,rendition-complement] [,word-terminator-code]
 [,character-set]
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### keyboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual keyboard from which input is to be read. The **keyboard-id** argument is the address of an unsigned longword that contains the keyboard identifier.

**Keyboard-id** is returned by SMG\$CREATE\_VIRTUAL\_KEYBOARD.

##### key-table-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the key table to be used for translating keystrokes. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

**Key-table-id** is returned by SMG\$CREATE\_KEY\_TABLE.

##### resultant-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

String into which SMG\$READ\_COMPOSED\_LINE writes the complete composed line. The **resultant-string** argument is the address of a descriptor pointing to the string in which the composed line is written.

**prompt-string**

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String used to prompt for the read operation. The **prompt-string** argument is the address of a descriptor pointing to the prompt string.

**resultant-length**

|               |                 |
|---------------|-----------------|
| OpenVMS usage | word_unsigned   |
| type          | word (unsigned) |
| access        | write only      |
| mechanism     | by reference    |

Receives the number of bytes read or the maximum length of **resultant-string**, whichever is less. The **resultant-length** argument is the address of an unsigned word into which SMG\$READ\_COMPOSED\_LINE writes the number of bytes read.

**display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Display identifier. The **display-id** argument is the address of an unsigned longword that contains the display identifier. This argument is optional only if you are not using the Screen Management Facility's output routines.

If you are using the Screen Management Facility input and output routines, this argument specifies the virtual display in which the input is to occur. The virtual display specified must be pasted to the same pasteboard as specified by **keyboard-id** and must not be occluded. You cannot accept input from an occluded area of the virtual display.

In the case of multiple virtual displays, each virtual display has an associated virtual cursor position. At the same time, there is a single physical cursor position corresponding to the current location of the physical cursor. If the **display-id** argument is specified, the read begins at the current virtual cursor position in the specified virtual display. If the display identifier is omitted, the read begins in the current physical cursor position. Note that the suppress leading code length of the **prompt-string** plus the suppress leading code length of the key entered is limited to the number of visible columns in the display.

---

**Note**


---

This virtual display must be pasted in column 1 and may not have any other virtual displays to its right. This restriction is necessary because otherwise any occurrence of Ctrl/R or Ctrl/U would blank out the entire line, including any output pasted to the right.

---

**flags**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |

## SMG\$READ\_COMPOSED\_LINE

mechanism           by reference

Optional bit mask that specifies enabled keys. The **flags** argument is the address of an unsigned longword that contains the flag. Valid values for **flags** are as follows:

|                  |                                                                       |
|------------------|-----------------------------------------------------------------------|
| 0                | Line editing is enabled and function keys (F6 to F14) cannot be used. |
| SMG\$M_FUNC_KEYS | Function keys (F6 to F14) may be used and line editing is disabled.   |
| SMG\$M_NOKEEP    | Lines entered in the recall buffer are not saved.                     |
| SMG\$M_NORECALL  | Line recall is disabled for this I/O only.                            |

Because the VMS terminal driver uses the F6 through F14 function keys for line editing on some terminals, you cannot have function keys and line editing enabled at the same time.

### initial-string

OpenVMS usage   char\_string  
type            character string  
access           read only  
mechanism       by descriptor

Optional string that contains the initial characters of the field. The **initial-string** argument is the address of a descriptor pointing to the string. The string is written to the display in the input area, as if it had been entered from the keyboard. It may be edited in the usual way (provided that **function-keys-flag** is not set).

### timeout

OpenVMS usage   longword\_signed  
type            longword (signed)  
access           read only  
mechanism       by reference

Optional timeout count. The **timeout** argument is the address of a signed longword containing the timeout count. If the **timeout** argument is specified, all characters entered before the timeout are returned in the buffer. If the **timeout** argument is omitted, characters are returned in the buffer until a terminator is encountered.

### rendition-set

OpenVMS usage   mask\_longword  
type            longword (unsigned)  
access           read only  
mechanism       by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|              |                                                      |
|--------------|------------------------------------------------------|
| SMG\$M_BLINK | Displays blinking characters.                        |
| SMG\$M_BOLD  | Displays characters in higher-than-normal intensity. |

- SMG\$M\_REVERSE      Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.
- SMG\$M\_UNDERLINE    Displays underlined characters.
- SMG\$M\_INVISIBLE    Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard.
- SMG\$M\_USER1 through SMG\$M\_USER8    Displays user-defined attributes.

The **display-id** argument must be specified when you use the **rendition-set** argument.

**rendition-complement**

- OpenVMS usage    mask\_longword
- type              longword (unsigned)
- access            read only
- mechanism        by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

**word-terminator-code**

- OpenVMS usage    word\_unsigned
- type              word (unsigned)
- access            write only
- mechanism        by reference

Key terminator code. The **word-terminator-code** argument is an unsigned word into which is written a code indicating what character or key terminated the read. Key terminator codes are of the form SMG\$K\_TRM\_keyname. The key names are listed in Table 3–1 in Chapter 3.

## SMG\$READ\_COMPOSED\_LINE

### character-set

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access read only  
mechanism by reference

Optional **character-set** is the address of an unsigned longword containing the character set of characters to be input. If this argument is omitted, the default character set of the specified virtual display is used. If no virtual display is used, the default character set indicated by the SMG\$DEFAULT\_CHARACTER\_SET logical name is used.

Valid values for this argument are :

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

Refer to Table 2-1 in Section 2.2.14 for the valid definitions of the SMG\$DEFAULT\_CHARACTER\_SET logical name.

### Description

SMG\$READ\_COMPOSED\_LINE reads a line composed of normal keystrokes and key equivalence strings as defined in the specified key table. Attributes of the key definition control whether the equivalence string is echoed and whether the read terminates with the defined keystroke. Normal keystrokes are always echoed.

A carriage return always terminates the read operation. If Ctrl/Z is typed and there is no definition for Ctrl/Z in the key definition table, "EXIT" is echoed and the read is terminated. If Ctrl/Z was the first character typed on the line, SMG\$\_EOF is returned. Otherwise, SMG\$\_EOF is returned on the next read operation. No other terminators are recognized except those specified as attributes in a key definition.

If the arrow keys and Ctrl/B are not defined, the previous lines read with the SMG\$READ\_xxxx routines can be recalled using the arrow keys. The number of lines saved for later recall depends upon the **recall-size** argument in SMG\$CREATE\_VIRTUAL\_KEYBOARD. The default is 20 lines.

Note that SMG\$READ\_COMPOSED\_LINE calls the SMG\$FLUSH\_BUFFER routine before performing the input operation. This ensures that the screen image is up to date at the time of the input operation. Display batching for both the pasteboard and virtual display must be off when you use SMG\$READ\_COMPOSED\_LINE.

When SMG\$C\_HANYU is specified for the **character-set** parameter, the leading codes of C2CB in hexadecimal will be included in the resultant string for each 4-byte character. The **resultant-length** does include the length of the leading codes. The users of Chinese Screen Management Facility should provide enough buffer size to hold the leading codes.

**Condition Values Returned**

|                                                            |                                                                                          |
|------------------------------------------------------------|------------------------------------------------------------------------------------------|
| SS\$_NORMAL                                                | Normal successful completion.                                                            |
| SS\$_ABORT                                                 | I/O operation aborted during execution (by SMG\$CANCEL_INPUT).                           |
| SS\$_CANCEL                                                | I/O operation canceled while queued (by SMG\$CANCEL_INPUT).                              |
| SMG\$_EOF                                                  | End of file.                                                                             |
| SMG\$_ILLBATFNC                                            | Input not allowed from a batched display.                                                |
| SMG\$_INVARG                                               | Invalid argument.                                                                        |
| SMG\$_INVCOL                                               | Invalid column. The read operation attempts to use a column outside the virtual display. |
| SMG\$_INVDIS_ID                                            | Invalid <b>display-id</b> .                                                              |
| SMG\$_INVKBD_ID                                            | Invalid <b>keyboard-id</b> .                                                             |
| SMG\$_INVKTB_ID                                            | Invalid <b>key-table-id</b> .                                                            |
| SMG\$_WRONUMARG                                            | Wrong number of arguments.                                                               |
| Any condition values returned by LIB\$COPY_R_DX.           |                                                                                          |
| Any condition values returned by \$GET (except RMS\$_EOF). |                                                                                          |
| Any condition values returned by \$QIOW.                   |                                                                                          |

## SMG\$READ\_FROM\_DISPLAY

---

### SMG\$READ\_FROM\_DISPLAY—Read Text from Display

The Read Text from Display routine reads a line of text from a virtual display.

#### Format

SMG\$READ\_FROM\_DISPLAY display-id ,resultant-string [,terminator-string] [,start-row] [,rendition-string]  
[,character-set-string]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display from which text is read. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### resultant-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

String into which SMG\$READ\_FROM\_DISPLAY writes the information read from the virtual display. The **resultant-string** argument is the address of a descriptor pointing to the string into which the string is written.

##### terminator-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String containing a terminator or terminators that end the backward search, thus determining the starting position of the returned string. The **terminator-string** argument is the address of a descriptor pointing to the string of terminators. If **terminator-string** is omitted, no back searching is performed; the returned string starts with the character at the current cursor position.

##### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |



The **start-row** argument is the address of a signed longword that contains the row of the virtual display to read from. This is an optional argument.

#### rendition-string

OpenVMS usage char\_string  
 type character string  
 access write only  
 mechanism by descriptor

Optional bit mask string that controls the video attributes. The **rendition-string** argument is the address of a descriptor pointing to the bit mask string. Each attribute set causes the corresponding attribute to be set for the corresponding byte in the text string in the display. The following attributes can be specified for each byte using the **rendition-string** argument:

SMG\$\_BLINK Displays blinking characters.  
 SMG\$\_BOLD Displays characters in higher-than-normal intensity.  
 SMG\$\_REVERSE Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.  
 SMG\$\_UNDERLINE Displays underlined characters.  
 SMG\$\_INVISIBLE Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard.

#### character-set-string

OpenVMS usage char\_string  
 type character string  
 access write only  
 mechanism by descriptor

Optional byte string that shows character set. The **character-set-string** argument is the address of a descriptor pointing to the character set string. Each byte shows the corresponding character set for corresponding byte in the text string in the display. The following values can be returned for each byte specified with **character-set-string** argument:

| Value               | Character Set         |
|---------------------|-----------------------|
| SMG\$_ASCII         | ASCII                 |
| SMG\$_SPEC_GRAPHICS | DEC Special Graphics  |
| SMG\$_HANYU         | DEC Hanyu             |
| SMG\$_UNKNOWN       | Unknown character set |

## Description

SMG\$READ\_FROM\_DISPLAY returns a string that contains some or all of the text on the current line of the specified virtual display. If the **terminator-string** argument is omitted, the contents of the current line (from the current column position to the rightmost column position) are returned. If the current column position is the right portion of a Chinese character, unknown character is returned for the position.

## SMG\$READ\_FROM\_DISPLAY

If some 4-byte character is output in the suppression mode, the leading codes of C2CB will be included in the resultant string for each 4-byte character. The users of the Chinese Screen Management Facility should notice that the resultant string length may become the double of the screen columns, and should provide enough buffer size to hold the leading codes. Also, the size of the rendition string and character set string should be double. The rendition and character set information of 4-byte character are duplicated for its leading code.

If the **start-row** argument is passed, the contents of line **start-row** from column 1 to the rightmost column are returned in **resultant-string**. If the **start-row** argument is passed, the **terminator-string** argument is ignored.

If you specify **terminator-string**, each character in it serves as a terminator for back searching, that is, the process of determining the first character position to be returned. If none of the specified terminators is encountered, the search is terminated at the first character position on the line.

Device-independent characters ( +, -, | ) are returned for drawn lines.

### Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .   |
| LIB\$_INVSTRDES | Invalid string descriptor.    |
| LIB\$_INSVIRMEM | Insufficient virtual memory.  |

### Example

```
C+
C This DEC Fortran example demonstrates the use of SMG$READ_FROM_DISPLAY.
C-

C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'
 CHARACTER*80 TEXT

C+
C Use SMG$CREATE_VIRTUAL_DISPLAY to create the virtual display
C and give it a border.
C-

 ROWS = 5
 COLUMNS = 60

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Create the pasteboard by calling SMG$CREATE_PASTEBOARD.
C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$PASTE_VIRTUAL_DISPLAY and SMG$PUT_LINE to paste
C the virtual display and put some text on line 2.
C-
```

## SMG\$READ\_FROM\_DISPLAY

```
STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 2, 10)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_LINE (DISPLAY1, ' ')
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_LINE (DISPLAY1,
1 'This is an example of using SMG$READ_FROM_DISPLAY.')
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$READ_FROM_DISPLAY to read line 2 from the virtual
C display, starting at column 22.
C-

STATUS = SMG$SET_CURSOR_ABS (DISPLAY1, 2, 22)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Search line 2 from column 22 to column 1 for the null string.
C Since no terminator will be supplied, no "back-searching" will take
C place. TEXT will be assigned the "value" of the line from
C column 22 to the rightmost column.
C-

STATUS = SMG$READ_FROM_DISPLAY (DISPLAY1, TEXT)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Put the line of text found into the virtual display at row 4,
C column 10 by calling SMG$SET_CURSOR_ABS and SMG$PUT_LINE.
C-

STATUS = SMG$SET_CURSOR_ABS (DISPLAY1, 4, 10)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_LINE (DISPLAY1, TEXT)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$SET_CURSOR_ABS to set the cursor back to line 2, column 22.
C-

STATUS = SMG$SET_CURSOR_ABS (DISPLAY1, 2, 22)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use SMG$READ_FROM_DISPLAY to search line 2 from column 22 to
C column 1 for an "f". Now, "back-searching" will take place.
C Starting at column 22, "back-track" to column 1 looking for "f".
C Text will then be assigned the "value" of the line from the
C present cursor position (where the "f" is, to the rightmost
C column.
C-

STATUS = SMG$READ_FROM_DISPLAY (DISPLAY1, TEXT, 'f')
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Put the line of text found into the virtual display at row 4, column 10.
C-

STATUS = SMG$SET_CURSOR_ABS (DISPLAY1, 5, 10)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_LINE (DISPLAY1, TEXT)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

END
```

The output generated by this FORTRAN program is shown in Figure SMG-36.

## SMG\$READ\_FROM\_DISPLAY

Figure SMG-36 Output Generated by FORTRAN Program Calling SMG\$READ\_FROM\_DISPLAY

```
This is an example of using SMG$READ_FROM_DISPLAY.
 using SMG$READ_FROM_DISPLAY.
f using SMG$READ_FROM_DISPLAY.
```

ZK-4134-GE

---

## SMG\$READ\_KEYSTROKE—Read a Single Character

The Read a Single Character routine reads a keystroke and returns that keystroke's terminator code.

### Format

```
SMG$READ_KEYSTROKE keyboard-id ,word-terminator-code [,prompt-string] [,timeout] [,display-id]
 [,rendition-set] [,rendition-complement]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### keyboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Keyboard identifier. The **keyboard-id** argument is an unsigned longword containing the identifier of the virtual keyboard from which to read.

You can create a virtual keyboard by calling the SMG\$CREATE\_VIRTUAL\_KEYBOARD routine.

#### word-terminator-code

|               |                 |
|---------------|-----------------|
| OpenVMS usage | word_unsigned   |
| type          | word (unsigned) |
| access        | write only      |
| mechanism     | by reference    |

Key terminator code. The **word-terminator-code** argument is an unsigned word into which is written a code indicating what character or key terminated the read. Key terminator codes are of the form SMG\$K\_TRM\_keyname. The key names are listed in Table 3-1 in Chapter 3.

#### prompt-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Prompt string. The **prompt-string** argument is an optional string used as the prompt for the read operation.

#### timeout

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

## SMG\$READ\_KEYSTROKE

Timeout count. The **timeout** argument is optional. If specified, any character typed before the timeout is returned in the buffer.

### **display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Display identifier. The optional **display-id** argument is the address of an unsigned longword that contains the identifier of the virtual display in which the read is to be performed. If the optional **prompt-string** argument is specified while there are multiple virtual displays paged, the **display-id** argument is required to determine in which virtual display the prompt string will be written. If the **prompt-string** argument is not specified, do not specify the **display-id** argument.

In the case of multiple virtual displays, each virtual display has an associated virtual cursor position. At the same time, there is a single physical cursor position corresponding to the current location of the physical cursor. If the **display-id** argument is specified, the read begins at the current virtual cursor position in the specified virtual display. If **display-id** is omitted, the read begins in the current physical cursor position. Note that the length of the **prompt-string** plus the key entered is limited to the number of visible columns in the display.

**rendition-set**

OpenVMS usage mask\_longword  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

SMG\$\_BLINK Displays blinking characters.  
 SMG\$\_BOLD Displays characters in higher-than-normal intensity.  
 SMG\$\_REVERSE Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display.  
 SMG\$\_UNDERLINE Displays underlined characters.  
 SMG\$\_USER1 through SMG\$\_USER8 Displays user-defined attributes.

The **display-id** argument must be specified when you use the **rendition-set** argument.

**rendition-complement**

OpenVMS usage mask\_longword  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

## SMG\$READ\_KEYSTROKE

### Description

SMG\$READ\_KEYSTROKE reads a keystroke from the specified virtual keyboard and returns the terminator code of that keystroke in the form SMG\$K\_TRM\_keyname. The keystroke entered to be read is not echoed on the screen. This keystroke may be any standard alphabetic character, any keypad or function key, or one of the directional arrows.

Note that display batching for both the pasteboard and the virtual display must be off when you use SMG\$READ\_KEYSTROKE.

You can enter all keys on the VT100, VT200-series, and VT300-series keyboards with the following exceptions:

- The Compose Character key on VT200-series and VT300-series keyboards
- The ESCAPE key
- The SHIFT keys
- The keys F1 through F5 on VT200-series and VT300-series keyboards

There are some keys and key definitions you can define, but Digital strongly suggests you avoid defining them. SMG\$ does not return an error when you use these keys and key definitions, but the definitions you assign to these key combinations are not executed unless you set your terminal in the following special ways at the DCL level:

- Ctrl/C, Ctrl/O, Ctrl/X, and F6 — To read these keys, you must first enter the DCL command SET TERMINAL/PASTHRU.
- Ctrl/T, Ctrl/Y — To read these keys, you must first enter either the DCL command SET TERMINAL/PASTHRU or SET NOCONTROL, or both.
- Ctrl/S, Ctrl/Q — To read these keys, you must first enter the DCL command SET TERMINAL/NOTTSYNC.

Digital does not recommend that you use these special terminal settings. The settings may cause unpredictable results if you do not understand all the implications of changing the default settings to give control to the terminal driver.

### Condition Values Returned

|                 |                                                                |
|-----------------|----------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                  |
| SS\$_ABORT      | I/O operation aborted during execution (by SMG\$CANCEL_INPUT). |
| SS\$_CANCEL     | I/O operation canceled while queued (by SMG\$CANCEL_INPUT).    |
| SS\$_xxx        | Any error from \$QIOW.                                         |
| SMG\$_EOF       | End of file.                                                   |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                    |
| SMG\$_INVKBD_ID | Invalid <b>keyboard-id</b> .                                   |
| SMG\$_WRONUMARG | Wrong number of arguments.                                     |



LIB\$\_xxx Any error from LIB\$SCOPY\_R\_DX.  
 RMS\$\_xxx Any error from \$GET (except RMS\$\_EOF).

## Examples

```

1. C+
 C This DEC Fortran example program demonstrates the use of
 C SMG$READ_KEYSTROKE.
 C-

 C+
 C This routine creates a virtual display and writes it to the pasteboard.
 C Data is placed in the virtual display using SMG$PUT_CHARS.
 C
 C First, include the SMG definitions. In particular, we want SMG$_BORDER.
 C-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'
 CHARACTER*3 TEXT
 CHARACTER*27 TEXT_OUTPUT

 C+
 C Use SMG$CREATE_VIRTUAL_DISPLAY to create a virtual
 C display with a border.
 C-

 ROWS = 7
 COLUMNS = 60

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Create the pasteboard using SMG$CREATE_PASTEBOARD.
 C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Use SMG$CREATE_VIRTUAL_KEYBOARD to create a virtual keyboard.
 C-

 STATUS = SMG$CREATE_VIRTUAL_KEYBOARD (KEYBOARD1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Using SMG$PASTE_VIRTUAL_DISPLAY, paste the virtual display
 C at row 3, column 9.
 C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 3, 9)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_LINE (DISPLAY1,
1 'Enter the character K after the >> prompt.')
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_LINE (DISPLAY1,
1 'This character will not be echoed as you type it.')
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_LINE (DISPLAY1,
1 'The terminal character equivalent of K is displayed.')
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

```

## SMG\$READ\_KEYSTROKE

```
STATUS = SMG$PUT_LINE (DISPLAY1, ' ')
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$READ_KEYSTROKE to read a keystroke from the virtual
C pasteboard.
C-

 STATUS = SMG$READ_KEYSTROKE (KEYBOARD1, TERM_CHAR, '>>', ,
1 DISPLAY1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_LINE (DISPLAY1, ' ')
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Use OTS$CVT_L_TI to convert the decimal value of TERM_CHAR to
C a decimal ASCII text string.
C-

 STATUS = OTS$CVT_L_TI (TERM_CHAR, TEXT)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 TEXT_OUTPUT = ' TERMINAL CHARACTER IS: ' // TEXT

C+
C Call SMG$PUT_LINE and SMG$PUT_CHARS to print the decimal
C ASCII text string.
C-

 STATUS = SMG$PUT_LINE (DISPLAY1, TEXT_OUTPUT)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1, TEXT, 7, 25)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 END
```

The output generated by this FORTRAN program is shown in Figure SMG-37.

**Figure SMG-37 Output Generated by FORTRAN Program Calling SMG\$READ\_KEYSTROKE**

```

Enter the character K after the >> prompt.
This character will not be echoed as you type it.
The terminal character equivalent of K is displayed.

>>

 TERMINAL CHARACTER IS: 107

```

ZK-4129-GE

```

2. 1 OPTION TYPE=EXPLICIT
 !+
 ! This VAX BASIC program demonstrates the use of
 ! SMG$READ_KEYSTROKE to read a keystroke from the terminal.
 !
 DECLARE LONG kb_id, ret_status, term_code, I, timer
 EXTERNAL SUB LIB$SIGNAL(LONG BY VALUE)
 EXTERNAL SUB LIB$STOP(LONG BY VALUE)
 EXTERNAL LONG CONSTANT SS$TIMEOUT
 %INCLUDE "$SMGDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"
 EXTERNAL LONG FUNCTION SMG$CREATE_VIRTUAL_KEYBOARD(LONG, STRING)
 EXTERNAL LONG FUNCTION SMG$DELETE_VIRTUAL_KEYBOARD(LONG)
 EXTERNAL LONG FUNCTION SMG$READ_KEYSTROKE(LONG, LONG, STRING, &
 LONG, LONG)

 !+
 ! Prompt the user for the timer value. A value of 0 will cause
 ! the type ahead buffer to be read.
 !-
 INPUT "Enter timer value (0 to read typeahead buffer): ";timer

 !+
 ! Establish a SMG connection to SYS$INPUT. Signal any unexpected
 ! errors.
 !-

```

## SMG\$READ\_KEYSTROKE

```
ret_status = SMG$CREATE_VIRTUAL_KEYBOARD(kb_id, "SYS$INPUT:")
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$SIGNAL(ret_status)
END IF

!+
! Read a keystroke, tell the user what we found.
!-

ret_status = SMG$READ_KEYSTROKE(kb_id, term_code, , timer,)
IF (ret_status <> SS$TIMEOUT) AND ((ret_status AND 1%) = 0%) THEN
 CALL LIB$SIGNAL(ret_status)
END IF

PRINT "term_code = ";term_code
SELECT term_code

 CASE 0 TO 31
 PRINT "You typed a control character"

 CASE 32 TO 127
 PRINT "You typed: ";CHR$(term_code)

 CASE SMG$K_TRM_PF1 TO SMG$K_TRM_PERIOD
 PRINT "You typed one of the keypad keys"

 CASE SMG$K_TRM_UP TO SMG$K_TRM_RIGHT
 PRINT "You typed one of the cursor positioning keys"

 CASE SMG$K_TRM_F6 TO SMG$K_TRM_F20
 PRINT "You typed one of the function keys"

 CASE SMG$K_TRM_FIND TO SMG$K_TRM_NEXT_SCREEN
 PRINT "You typed one of the editing keys"

 CASE SMG$K_TRM_TIMEOUT
 PRINT "You did not type a key fast enough"

 CASE ELSE
 PRINT "I'm not sure what key you typed"

END SELECT

!+
! Close the connection to SYS$INPUT, and signal any errors.
!-

ret_status = SMG$DELETE_VIRTUAL_KEYBOARD(kb_id)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$SIGNAL(ret_status)
END IF

END
```

This BASIC program reads a key and returns the **word-terminator-code** and the name of the keystroke entered. One sample of the commands entered and the output generated by this program is as follows:

```
$ RUN READ_KEY
Enter the timer value (0 to read type-ahead buffer): ? 9
term_code = 100
You typed: d
```

Note that in this example, the user entered the keystroke “d” following the first prompt. The keystroke entered was not echoed.

---

## SMG\$READ\_STRING—Read String

The Read String routine reads a string from a virtual keyboard.

### Format

```
SMG$READ_STRING keyboard-id ,resultant-string [,prompt-string] [,maximum-length] [,modifiers]
 [,timeout] [,terminator-set] [,resultant-length] [,word-terminator-code] [,display-id]
 [,initial-string] [,rendition-set] [,rendition-complement] [,terminator-string]
 [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### keyboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual keyboard from which input is to be read. The **keyboard-id** argument is the address of an unsigned longword that contains the keyboard identifier.

**Keyboard-id** is returned by SMG\$CREATE\_VIRTUAL\_KEYBOARD.

#### resultant-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

String into which the input line is written. The **resultant-string** argument is the address of a descriptor pointing to the string into which the text is written.

#### prompt-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String used to prompt for the read operation. The **prompt-string** argument is the address of a descriptor pointing to the prompt string.

#### maximum-length

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

## SMG\$READ\_STRING

Specifies the maximum suppress leading code input byte length to be read. The **maximum-length** argument is the address of a signed longword that contains the maximum suppress leading code input byte length to be read. The maximum valid value for this argument is 512. If omitted, 512 is the default.

Note that as a result of truncation, the suppress leading code length of the return string may be one byte shorter than **maximum-length** when the last byte is the left portion of Chinese character.

### modifiers

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask that specifies optional behavior. The **modifiers** argument is the address of an unsigned longword that contains the flag.

Valid values for **modifiers** are as follows:

|                  |                                                                                                  |
|------------------|--------------------------------------------------------------------------------------------------|
| TRM\$M_TM_CVTLOW | Converts lowercase characters to uppercase. Only valid for ASCII and supplemental character set. |
| TRM\$M_TM_NOECHO | Characters entered are not echoed on the screen.                                                 |
| TRM\$M_TM_PURGE  | Type-ahead buffer purged before read is done.                                                    |
| TRM\$M_TM_       | Terminator is not echoed.                                                                        |
| TRMNOECHO        |                                                                                                  |
| TRM\$M_TM_NOEDIT | Advanced editing is disabled.                                                                    |
| TRM\$M_TM_       | Line recall is disabled.                                                                         |
| NORECALL         |                                                                                                  |

The TRM\$ symbols are defined by the \$TRMDEF macro/module in Digital-supplied system symbol libraries.

### timeout

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the number of seconds allowed between the time the prompt is issued and the completion of the input operation. The **timeout** argument is the address of a signed longword that contains the timeout value.

If **timeout** is specified, all characters typed before the expiration time or until a terminate key is entered are returned in **resultant-string**. If omitted, the input operation remains active until a terminator is typed.

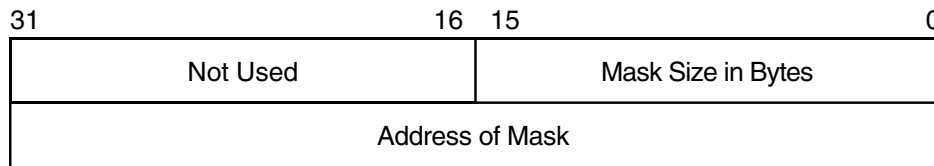
### terminator-set

|               |                             |
|---------------|-----------------------------|
| OpenVMS usage | unspecified                 |
| type          | unspecified                 |
| access        | read only                   |
| mechanism     | by descriptor, fixed length |

Either a mask that specifies which characters are to be treated as terminators (short form) or a descriptor pointing to such a mask (long form). The **terminator-set** argument is the address of a descriptor pointing to the mask.

If you want to use terminators with ASCII values in the range 0 to 31, use the short form, which consists of two longwords. The first longword is zero, and the second longword is a terminator character bit mask. You create this mask by setting the bit that corresponds to the ASCII value of the desired terminator. For example, to specify that Ctrl/A (ASCII value 1) is a terminator, you set bit 1 in the **terminator-set** mask.

If you want to use terminators with ASCII values outside the range 0 to 31, use the long form. First create a descriptor of this form:



ZK-2004-GE

The mask itself has the same format as that of the short form; however, the long form allows the use of a more comprehensive set of terminator characters. For example, a mask size of 16 bytes allows any 7-bit ASCII character to be set as a terminator, while a mask size of 32 bytes allows any 8-bit character to be set as a terminator. Any mask size between 1 and 32 bytes is acceptable.

If the terminator mask is all zeros, there are no specified terminators. In that case, the read terminates when the number of bytes specified in **maximum-length** has been transferred or when timeout is reached.

If the **terminator-set** argument is omitted, the set of terminators is the OpenVMS default terminator set.

#### resultant-length

OpenVMS usage word\_unsigned  
 type word (unsigned)  
 access write only  
 mechanism by reference

Receives the number of bytes read or the maximum size of **resultant-string**, whichever is less. The **resultant-length** argument is the address of an unsigned word into which is written the number of bytes read or the maximum size.

#### word-terminator-code

OpenVMS usage word\_unsigned  
 type word (unsigned)  
 access write only  
 mechanism by reference

Key terminator code. The **word-terminator-code** argument is an unsigned word into which is written a code indicating what character or key terminated the read. Key terminator codes are of the form SMG\$K\_TRM\_keyname. The keynames are listed in Table 3-1 in Chapter 3.

#### display-id

OpenVMS usage identifier  
 type longword (unsigned)  
 access read only  
 mechanism by reference

## SMG\$READ\_STRING

Display identifier. The **display-id** argument is the address of an unsigned longword that contains the display identifier. This argument is optional only if you are not using the Screen Management Facility's output routines.

If you are using the Screen Management Facility input and output routines, this argument specifies the virtual display in which the input is to occur. The virtual display specified must be pasted to the same pasteboard as specified by **keyboard-id** and must not be occluded. You cannot accept input from an occluded area of the virtual display.

In the case of multiple virtual displays, each virtual display has an associated virtual cursor position. At the same time, there is a single physical cursor position corresponding to the current location of the physical cursor. If the **display-id** argument is specified, the read begins at the current virtual cursor position in the specified virtual display. If **display-id** is omitted, the read begins in the current physical cursor position. Note that the suppress leading code length of the **prompt-string** plus the suppress leading code length of key entered is limited to the number of visible columns in the display.

---

### Note

---

This virtual display must be pasted in column 1 and may not have any other virtual displays to its right. This restriction applies because otherwise the occurrence of a Ctrl/R or Ctrl/U would cause the entire line to be blanked, including any output to the right.

---

### initial-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Initial character string. The **initial-string** argument is the address of a descriptor pointing to the optional string that contains the initial characters of the field. The string is written to the display in the input area, as if it had been entered from the keyboard. It may be edited in the usual way (provided that TRM\$M\_TM\_NOEDIT is not set). Its suppress leading code length must be shorter than **maximum-length**, otherwise SMG\$READ\_STRING will return immediately without accepting any keyboard input.

### rendition-set

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|              |                                                               |
|--------------|---------------------------------------------------------------|
| SMG\$M_BLINK | Displays blinking characters.                                 |
| SMG\$M_BOLD  | Displays characters in higher-than-normal intensity (bolded). |



|                                         |                                                                                                                           |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| SMG\$M_REVERSE                          | Displays characters in reverse video; that is, using the opposite default rendition of the virtual display.               |
| SMG\$M_UNDERLINE                        | Displays underlined characters.                                                                                           |
| SMG\$M_INVISIBLE                        | Specifies invisible characters; that is, the characters exist in the virtual display but do not appear on the pasteboard. |
| SMG\$M_USER1<br>through<br>SMG\$M_USER8 | Displays user-defined attributes.                                                                                         |

The **display-id** argument must be specified when you use the **rendition-set** argument.

#### rendition-complement

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All of the attributes that can be specified with the **rendition-set** argument can be complemented with **rendition-complement**. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display in which the read is done. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of current setting |
| 1   | 1          | Attribute off                                  |

#### terminator-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

Characters that terminated I/O. The optional **terminator-string** argument is the address of a descriptor pointing to the character string containing the terminating characters. **Terminator-string** returns the actual terminating characters, not the key that was pressed to terminate the I/O.

## SMG\$READ\_STRING

### character-set

OpenVMS usage unsigned\_longword  
type longowrd (unsigned)  
access read only  
mechanism by reference

Specifies character set of the text to be input. The optional **character-set** argument is the address of a unsigned longword containing character set of characters to be input. If this argument is omitted, the default character set of the specified virtual display is used. If no virtual display is used, the default character set indicated by the SMG\$DEFAULT\_CHARACTER\_SET logical name is used.

Valid values for **character-set** are as follows:

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

Refer to Table 2–1 in Section 2.2.14 for the valid definitions of the SMG\$DEFAULT\_CHARACTER\_SET logical name.

### Description

SMG\$READ\_STRING returns a string of characters read from a virtual keyboard. Note that display batching for both the pasteboard and the virtual display must be off when you use SMG\$READ\_STRING.

The text read by SMG\$READ\_STRING is saved for later recall with SMG\$READ\_COMPOSED\_LINE. Rendition applies to both the prompt (if there is one) and echoing of input characters.

When SMG\$C\_HANYU is specified in **character-set**, the leading codes of C2CB in hexadecimal will be included in the resultant string for each 4-byte character. The **resultant-string** does include the length of the leading codes. The users of Chinese Screen Management Facility should provide enough buffer size to hold the leading codes. Please note that the **maximum-length** specifies the maximum suppress leading code input length to be read, and that users should not include the length of the leading codes in the **maximum-length**.

### Condition Values Returned

|                 |                                                                |
|-----------------|----------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                  |
| SS\$_ABORT      | I/O operation aborted during execution (by SMG\$CANCEL_INPUT). |
| SS\$_CANCEL     | I/O operation canceled while queued (by SMG\$CANCEL_INPUT).    |
| SMG\$_EOF       | End of file.                                                   |
| SMG\$_ILLBATFNC | Input not allowed from a batched display.                      |
| SMG\$_INVARG    | Invalid argument.                                              |

|                 |                                                               |
|-----------------|---------------------------------------------------------------|
| SMG\$_INVCOL    | Invalid column. The input occurs outside the virtual display. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                   |
| SMG\$_INVKBD_ID | Invalid <b>keyboard-id</b> .                                  |
| SMG\$_INVKTB_ID | Invalid <b>key-table-id</b> .                                 |
| SMG\$_INVMAXLEN | Maximum length specified was greater than 512.                |
| SMG\$_WRONUMARG | Wrong number of arguments.                                    |

Any condition values returned by LIB\$COPY\_R\_DX.

Any condition values returned by \$GET (except RMS\$\_EOF).

Any condition values returned by \$QIOW.

## Examples

```

1. 1 OPTION TYPE=EXPLICIT

 !+
 ! This VAX BASIC program demonstrates the use of
 ! SMG$READ_STRING to read either a string,
 ! a control key, or a keypad key.
 !
 DECLARE LONG KB_ID, RET_STATUS, STR_LEN, TERM_CODE, MODIFIER, I, &
 TIMER
 DECLARE STRING DATA_STR, TERM_SET
 EXTERNAL LONG CONSTANT IO$M_TIMED
 EXTERNAL LONG CONSTANT IO$M_NOECHO
 EXTERNAL LONG CONSTANT IO$M_NOFILTR
 EXTERNAL SUB LIB$SIGNAL(LONG BY VALUE)
 EXTERNAL SUB LIB$STOP(LONG BY VALUE)
 EXTERNAL LONG CONSTANT SS$TIMEOUT
 EXTERNAL LONG FUNCTION SMG$CREATE_VIRTUAL_KEYBOARD(LONG, STRING)
 EXTERNAL LONG FUNCTION SMG$DELETE_VIRTUAL_KEYBOARD(LONG)
 EXTERNAL LONG FUNCTION SMG$READ_STRING(LONG, STRING, STRING, &
 LONG, LONG, LONG, STRING, LONG, LONG)
 %INCLUDE "$SMGDEF" %FROM %LIBRARY "SYS$LIBRARY: BASIC$STARLET"

 !+
 ! Prompt the user for the timer value. A value of 0 will cause
 ! the type-ahead buffer to be read.
 !-

 INPUT "Enter timer value (0 to read type-ahead buffer): ";TIMER

 !+
 ! Tell SMG to use the timer value
 !-

 MODIFIER = IO$M_TIMED

 !+
 ! Establish a SMG connection to SYS$INPUT. Signal any unexpected
 ! errors.
 !-

 RET_STATUS = SMG$CREATE_VIRTUAL_KEYBOARD(KB_ID, "SYS$INPUT:")
 IF (RET_STATUS AND 1%) = 0% THEN
 CALL LIB$SIGNAL(RET_STATUS)
 END IF

```

## SMG\$READ\_STRING

```
!+
! Tell SMG to use any keystroke except a letter or number
! as a terminator to the input and perform the read.
! Signal any error except SS$TIMEOUT
!-

TERM_SET = STRING$(4%, -1%) + STRING$(12%, 0%)
RET_STATUS = SMG$READ_STRING(KB_ID, DATA_STR, , , &
 MODIFIER, TIMER, TERM_SET, &
 STR_LEN, TERM_CODE)
IF (RET_STATUS <> SS$TIMEOUT) AND ((RET_STATUS AND 1%) = 0%) THEN
 CALL LIB$SIGNAL(RET_STATUS)
END IF

!+
! All the data should come back as a terminator code, since any
! character can be a terminator.
!-

PRINT "data string = ";LEFT(DATA_STR, STR_LEN)
PRINT "term_code = ";TERM_CODE
SELECT TERM_CODE
 CASE 0 TO 31
 PRINT "You typed a control character"
 CASE 32 TO 127
 PRINT "You typed: ";CHR$(TERM_CODE)
 CASE SMG$K_TRM_PF1 TO SMG$K_TRM_PERIOD
 PRINT "You typed one of the keypad keys"
 CASE SMG$K_TRM_UP TO SMG$K_TRM_RIGHT
 PRINT "You typed one of the cursor positioning keys"
 CASE SMG$K_TRM_F6 TO SMG$K_TRM_F20
 PRINT "You typed one of the function keys"
 CASE SMG$K_TRM_E1 TO SMG$K_TRM_E6
 PRINT "You typed one of the editing keys"
 CASE SMG$K_TRM_TIMEOUT
 PRINT "You did not type a key fast enough"
 CASE ELSE
 PRINT "I'm not sure what key you typed"
END SELECT

!+
! Close the connection to SYS$INPUT, and signal any errors.
!-

RET_STATUS = SMG$DELETE_VIRTUAL_KEYBOARD(KB_ID)
IF (RET_STATUS AND 1%) = 0% THEN
 CALL LIB$SIGNAL(RET_STATUS)
END IF

END
```

This BASIC example program demonstrates the use of SMG\$READ\_STRING. One sample of the output generated by this program is as follows:

```
$ RUN READ_STRING
Enter timer value (0 to read type-ahead buffer): ? 5
d
data string = d
term_code = 13
You typed a control character
```

```

2. C+
 C This DEC Fortran example program demonstrates how to use
 C SMG$READ_STRING.
 C
 C This routine creates a virtual display and writes it to the pasteboard.
 C Data is placed in the virtual display using SMG$PUT_CHARS.
 C-

 C+
 C Include the SMG definitions. In particular, we want SMG$M_BORDER.
 C-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'
 CHARACTER*20 TEXT

 C+
 C Create a virtual display with a border using SMG$CREATE_VIRTUAL_DISPLAY.
 C-

 ROWS = 7
 COLUMNS = 50

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Use SMG$CREATE_PASTEBOARD to create the pasteboard.
 C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Create a virtual keyboard by calling SMG$CREATE_VIRTUAL_KEYBOARD.
 C-

 STATUS = SMG$CREATE_VIRTUAL_KEYBOARD (KEYBOARD1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Use SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display
 C at row 3, column 9.
 C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 3, 9)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 C+
 C Read a string from the virtual pasteboard using SMG$READ_STRING.
 C-

 STATUS = SMG$READ_STRING (KEYBOARD1,
1 TEXT, 'prompt', 20, , , , , , DISPLAY1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 END

```

The output generated by this FORTRAN program before the call to SMG\$READ\_STRING is shown in Figure SMG-38. The program is waiting for input. The cursor immediately follows the word "prompt."

## SMG\$READ\_STRING

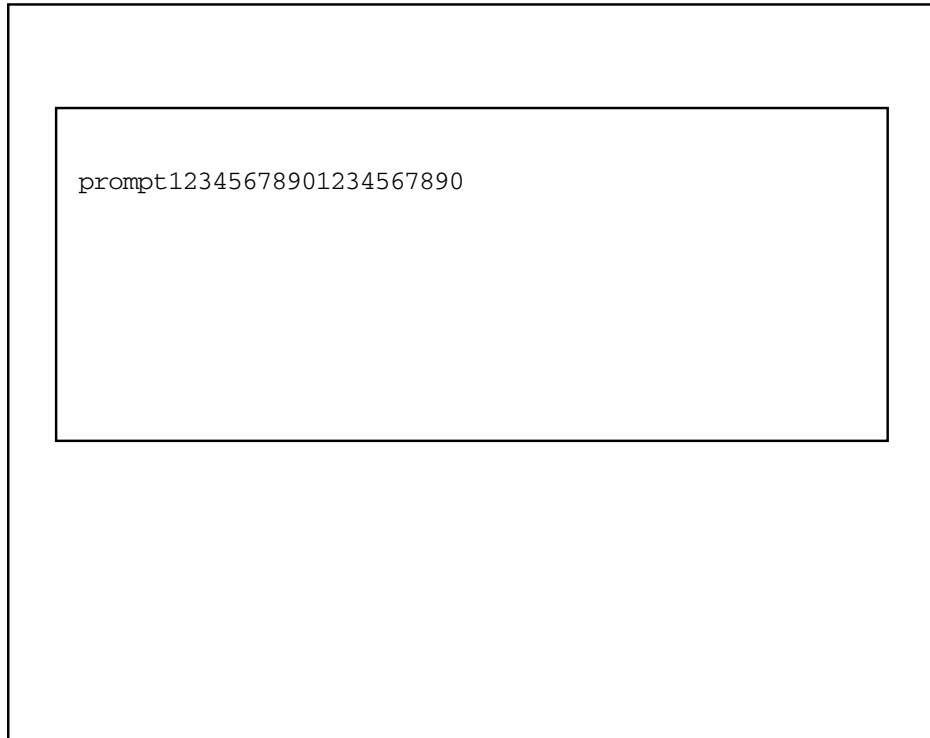
Figure SMG-38 Output Generated Before the Call to SMG\$READ\_STRING



ZK-4138-GE

The output generated after the call to SMG\$READ\_STRING is shown in Figure SMG-39.

Figure SMG-39 Output Generated After the Call to SMG\$READ\_STRING



ZK-4140-GE

## SMG\$REMOVE\_LINE

---

### SMG\$REMOVE\_LINE—Remove a Line from a Virtual Display

The Remove a Line from a Virtual Display routine removes a line from a specified virtual display that was drawn with the SMG\$DRAW\_LINE or SMG\$DRAW\_RECTANGLE routines.

#### Format

SMG\$REMOVE\_LINE display-id ,start-row ,start-column ,end-row ,end-column

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display from which the line is to be removed. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### **start-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row at which to begin removing the line. The **start-row** argument is the address of a signed longword that contains the row number.



**start-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column at which to begin removing the line. The **start-column** argument is the address of a signed longword that contains the column number.

In the case that the line is horizontal and the **start-column** begins on the right portion of a Chinese character, the left portion of the character will become an undefined character. If the line is vertical and the **start-column** is on the left or right portion of a Chinese characters for each row, the remaining portions of the characters become undefined characters.

**end-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row at which the line to be removed ends. The **end-row** argument is the address of a signed longword that contains the row number.

**end-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column at which the line to be removed ends. The **end-column** argument is the address of a signed longword that contains the column number.

In case that the line is horizontal and the **end-column** ends on the left portion of a Chinese character, the right portion of the character will become an undefined character. If the line is vertical and the **end-column** is on the left or right portion of a Chinese characters for each row, the remaining portions of the characters will become undefined characters.

## SMG\$REMOVE\_LINE

### Description

SMG\$REMOVE\_LINE removes a line drawn with SMG\$DRAW\_LINE or SMG\$DRAW\_RECTANGLE from a specified starting row and column to a specified ending row and column.

This routine erases the line you specify but preserves the line-drawing characters at any line intersection. (The line-drawing characters are the terminal's line-drawing character set. If that is not available, the characters +, -, and | are used.)

In case that the Chinese language character set is used, and the drawn line is on the left or right portion of the Chinese characters, the remaining portions of the characters will become undefined characters.

### Condition Values Returned

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                               |
| SMG\$_INVCOL    | Invalid column number. The specified column is outside the virtual display. |
| SMG\$_INVROW    | Invalid row number. The specified row is outside the virtual display.       |
| SMG\$_DIALINNOT | Diagonal line not allowed.                                                  |
| SMG\$_WRONUMARG | Wrong number of arguments.                                                  |

---

## SMG\$REPAINT\_SCREEN—Repaint Current Pasteboard

The Repaint Current Pasteboard routine repaints the specified pasteboard after non-SMG\$ I/O has occurred.

### Format

SMG\$REPAINT\_SCREEN pasteboard-id

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### ARGUMENT

|                      |                     |
|----------------------|---------------------|
| <b>pasteboard-id</b> |                     |
| OpenVMS usage        | identifier          |
| type                 | longword (unsigned) |
| access               | read only           |
| mechanism            | by reference        |

Specifies the pasteboard to be repainted. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

### Description

SMG\$REPAINT\_SCREEN repaints the specified pasteboard. It is intended to be used when some outside agent (for example, a broadcast message) has disrupted the pasteboard.

### Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SMG\$_WRONUMARG | Wrong number of arguments.     |

### Example

```
C+
C This DEC Fortran example program demonstrates
C the use of SMG$REPAINT_SCREEN.
C-
 IMPLICIT INTEGER (A-Z)
```

## SMG\$REPAINT\_SCREEN

```
C+
C Create the virtual display by calling
C SMG$CREATE_VIRTUAL_DISPLAY. To create
C a border, we set BORDER = 1. No border
C would be BORDER = 0.
C-
 INCLUDE '($SMGDEF)'
 ROWS = 3
 COLUMNS = 50
 BORDER = 1

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Create the pasteboard using SMG$CREATE_PASTEBOARD.
C-
 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Put data in the virtual display by calling SMG$PUT_CHARS.
C-
 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This virtual display has 3 rows and 50 columns.', 1, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This is a bordered virtual display.', 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 3, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-
 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

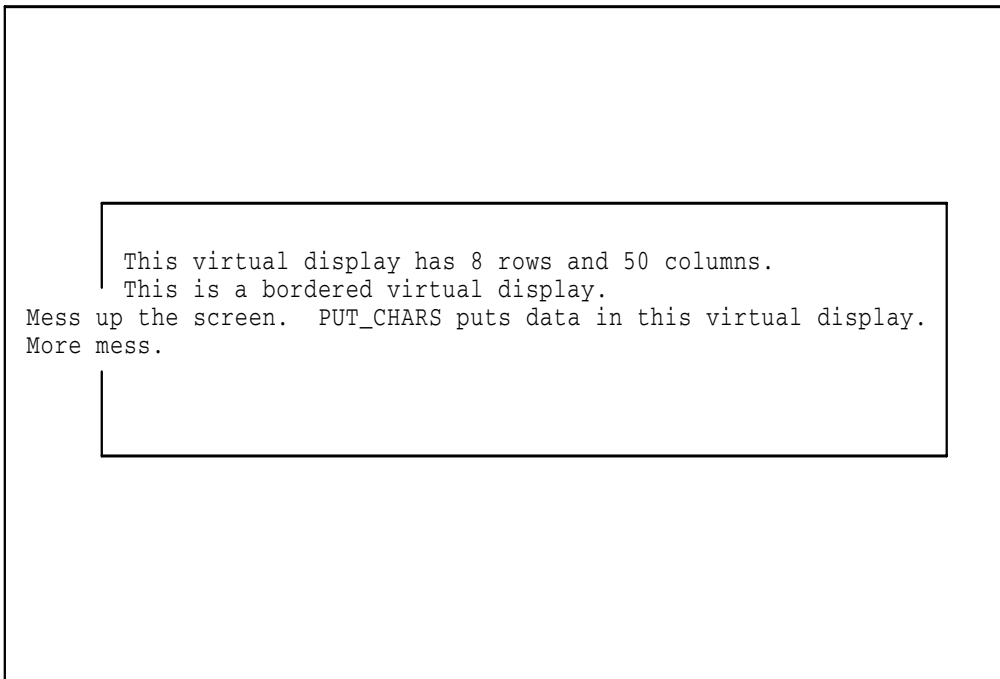
C+
C Mess up the screen with some FORTRAN output.
C-
 WRITE (6,*) 'Mess up the screen.'
 WRITE (6,*) 'More mess.'

C+
C Call SMG$REPAINT_SCREEN to repaint the screen.
C-
 STATUS = SMG$REPAINT_SCREEN (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 END
```

The output generated by this FORTRAN program before the call to SMG\$REPAINT\_SCREEN is shown in Figure SMG-40.

Figure SMG-40 Output Generated by FORTRAN Program Calling  
SMG\$REPAINT\_SCREEN

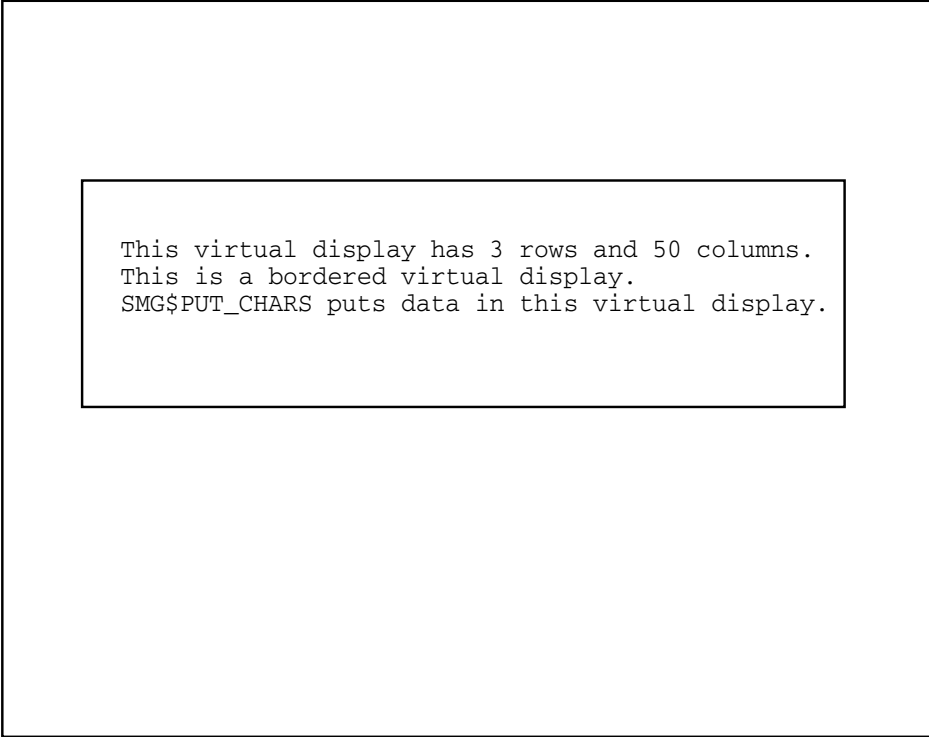


ZK-4136-GE

The output generated after the call to SMG\$REPAINT\_SCREEN is shown in  
Figure SMG-41.

## SMG\$REPAINT\_SCREEN

Figure SMG-41 Output Generated by FORTRAN Program Calling  
SMG\$REPAINT\_SCREEN



This virtual display has 3 rows and 50 columns.  
This is a bordered virtual display.  
SMG\$PUT\_CHARS puts data in this virtual display.

ZK-4137-GE

---

## SMG\$REPASTE\_VIRTUAL\_DISPLAY—Repaste Virtual Display

The Repaste Virtual Display routine moves a virtual display to a new position on the pasteboard. The pasting order is not preserved.

### Format

```
SMG$REPASTE_VIRTUAL_DISPLAY display-id ,pasteboard-id ,pasteboard-row ,pasteboard-column
 [,top-display-id]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### **display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display to be repasted. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### **pasteboard-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard on which the display is repasted. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

## SMG\$REPASTE\_VIRTUAL\_DISPLAY

### **pasteboard-row**

OpenVMS usage longword\_signed  
type longword (signed)  
access read only  
mechanism by reference

Specifies the pasteboard row that is to contain row 1 of the specified virtual display. The **pasteboard-row** argument is the address of a signed longword that contains the pasteboard row.

### **pasteboard-column**

OpenVMS usage longword\_signed  
type longword (signed)  
access read only  
mechanism by reference

Specifies the pasteboard column that is to contain column 1 of the specified virtual display. The **pasteboard-column** argument is the address of a signed longword that contains the pasteboard column.

### **top-display-id**

OpenVMS usage identifier  
type longword (unsigned)  
access read only  
mechanism by reference

Optional identifier of the virtual display under which **display-id** will be pasted. The **top-display-id** argument is the address of an unsigned longword containing the identifier of this virtual display. Note that the virtual display specified by **top-display-id** must already be pasted.

## Description

SMG\$REPASTE\_VIRTUAL\_DISPLAY lets you move a virtual display to a new position on its pasteboard. This routine calls SMG\$UNPASTE\_VIRTUAL\_DISPLAY and SMG\$PASTE\_VIRTUAL\_DISPLAY. Note that this changes the pasting order. The unpasting and repasting operations use the SMG\$BEGIN\_PASTEBOARD\_UPDATE and SMG\$END\_PASTEBOARD\_UPDATE routines; thus, there is no effect on the screen until the repasting operation is complete.

Note that this routine may cause the virtual display to be at the top of the pasting order. To move a virtual display without changing its pasting order, use SMG\$MOVE\_VIRTUAL\_DISPLAY. If the optional argument **top-display-id** is specified, SMG\$REPASTE\_VIRTUAL\_DISPLAY pastes the virtual display being repasted under the virtual display specified by **top-display-id**. In this case, the virtual display specified by **top-display-id** must already be pasted.

## Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .    |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SMG\$_WRONUMARG | Wrong number of arguments.     |



## Example

```

C+
C This DEC Fortran example program demonstrates the use of
C SMG$REPASTE_VIRTUAL_DISPLAY and SMG$MOVE_VIRTUAL_DISPLAY.
C-

 IMPLICIT INTEGER (A-Z)

C+
C Include the SMG definitions. In particular, we want SMG$M_BORDER.
C-

 INCLUDE '($SMGDEF)'

C+
C Create a virtual display with a border by calling
C SMG$CREATE_VIRTUAL_DISPLAY.
C-

 ROWS = 3
 COLUMNS = 50

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$CREATE_PASTEBOARD to create the pasteboard.
C-

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Put data in the virtual display using SMG$PUT_CHARS.
C-

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This virtual display has 3 rows and 50 columns.', 1, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' This is a bordered virtual display.', 2, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

 STATUS = SMG$PUT_CHARS (DISPLAY1,
1 ' SMG$PUT_CHARS puts data in this virtual display.', 3, 1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$PASTE_VIRTUAL_DISPLAY to paste the virtual display.
C-

 STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Move the virtual display by calling SMG$MOVE_VIRTUAL_DISPLAY.
C-

 STATUS = SMG$MOVE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 10, 5)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C+
C Call SMG$REPASTE_VIRTUAL_DISPLAY to repaste the
C original virtual display as it was.
C-

 STATUS = SMG$REPASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 15)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

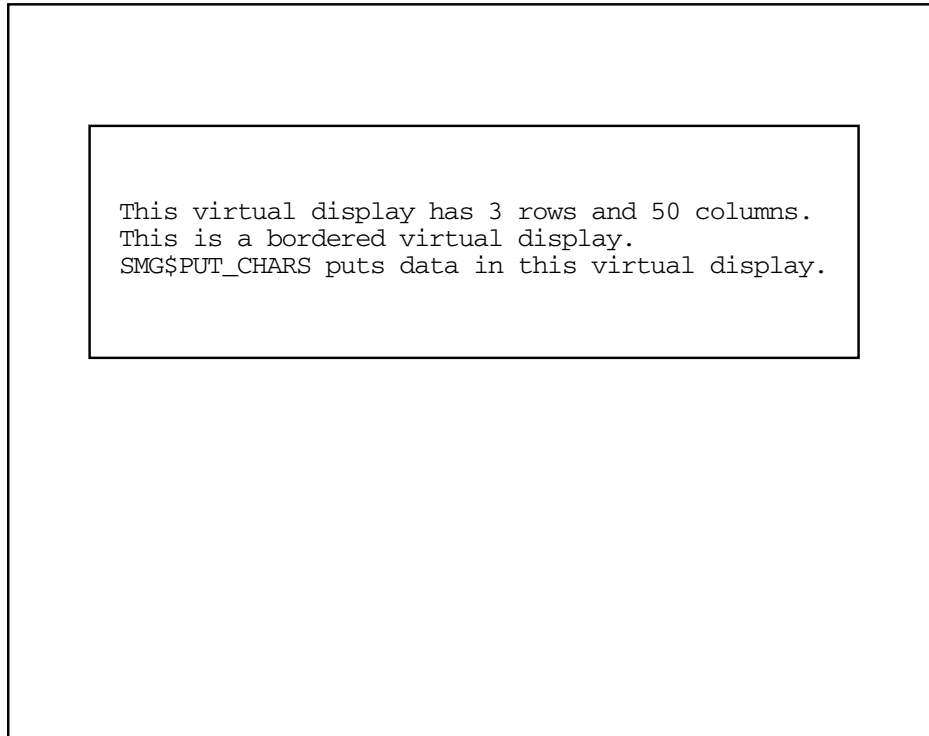
```

## SMG\$REPASTE\_VIRTUAL\_DISPLAY

END

The output generated by this FORTRAN program before the call to SMG\$MOVE\_VIRTUAL\_DISPLAY is shown in Figure SMG-42.

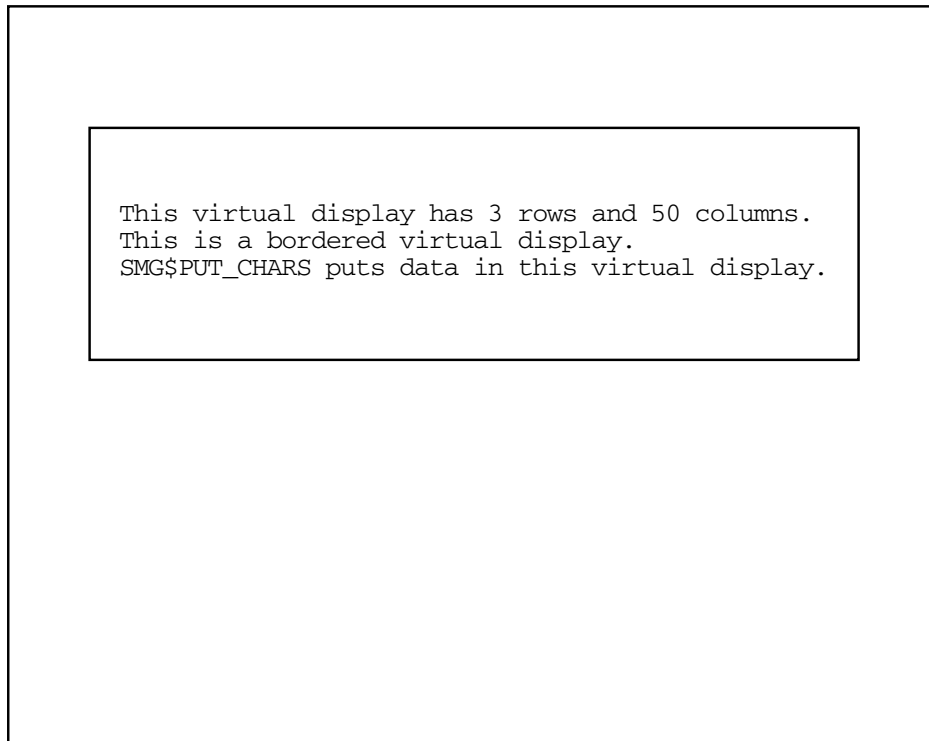
**Figure SMG-42 Output Before the Call to SMG\$MOVE\_VIRTUAL\_DISPLAY**



ZK-4139-GE

After the call to SMG\$MOVE\_VIRTUAL\_DISPLAY, the output is that shown in Figure SMG-43.

Figure SMG-43 Output Displayed After the Call to SMG\$MOVE\_VIRTUAL\_DISPLAY

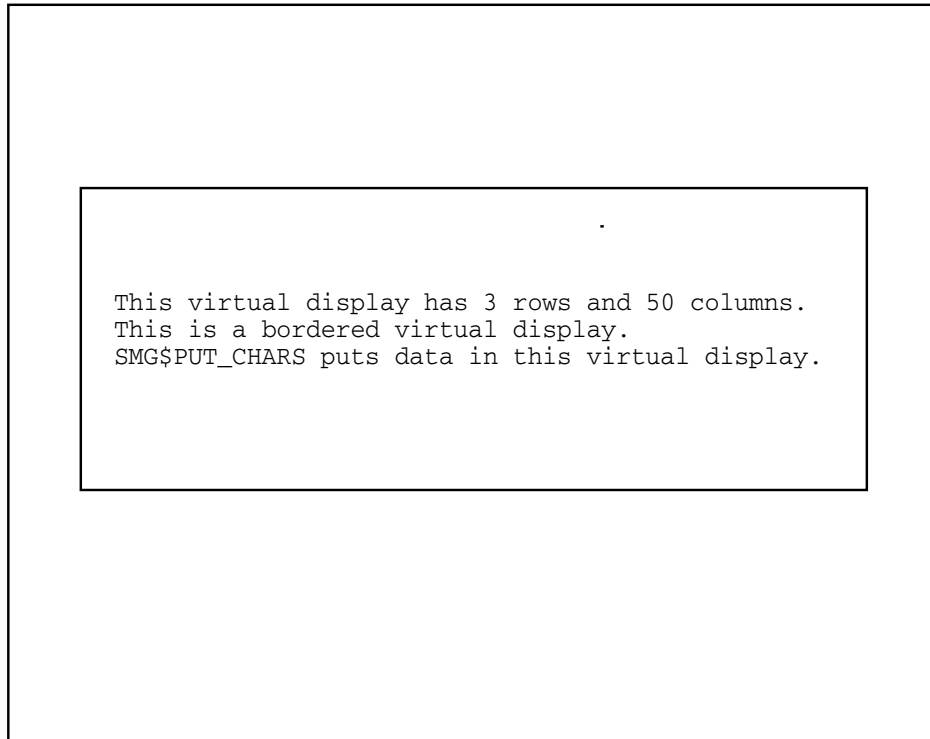


ZK-4141-GE

Figure SMG-44 shows the final output displayed after the call to SMG\$REPASTE\_VIRTUAL\_DISPLAY.

## SMG\$REPASTE\_VIRTUAL\_DISPLAY

Figure SMG-44 Output Displayed After the Call to SMG\$REPASTE\_VIRTUAL\_DISPLAY



ZK-4130-GE

---

## SMG\$REPLACE\_INPUT\_LINE—Replace Input Line

The Replace Input Line routine replaces the specified lines in the recall buffer with the specified string.

### Format

SMG\$REPLACE\_INPUT\_LINE keyboard-id [,replace-string] [,line-count] [,flags] [,character-set]

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### keyboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Keyboard identifier. The **keyboard-id** argument is the address of an unsigned longword containing the identifier of the virtual keyboard from which to read.

You create a virtual keyboard by calling the SMG\$CREATE\_VIRTUAL\_KEYBOARD routine.

#### replace-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String that contains the line to be entered into the recall buffer. The **replace-string** argument is the address of a descriptor pointing to this string. The default is a null string, which removes the last line entered.

## SMG\$REPLACE\_INPUT\_LINE

### line-count

OpenVMS usage byte\_unsigned  
type byte (unsigned)  
access read only  
mechanism by reference

Number of lines to be replaced. The **line-count** argument is the address of an unsigned byte containing the number of lines to be replaced with **replace-string**. The default value for the **line-count** argument is 1 (the last line entered).

### flags

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Specifies optional behavior. The **flags** argument is the address of an unsigned longword that contains the flag. Valid values are as follows:

SMG\$M\_KEEP\_CONTENTS                      The string specified by **replace-string** replaces the existing string specified by **line-count**. By using this flag in a loop, the line recall buffer can be preloaded with information by the application.

### character-set

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access read only  
mechanism by reference

Character set specifier. The optional **character-set** argument is the address of an unsigned longword that specifies the character set for the **replace-string**. If this argument is omitted, the default character set value is UNKNOWN.

| Value                | Character Set        |
|----------------------|----------------------|
| SMG\$C_ASCII         | ASCII                |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics |
| SMG\$C_HANYU         | DEC Hanyu            |

## Description

SMG\$REPLACE\_INPUT\_LINE replaces the requested lines in the recall buffer with the specified string. The remaining (**line-count**-1) lines are deleted. This routine is intended to aid in processing line continuations.

## Condition Values Returned

SS\$\_NORMAL                      Normal successful completion.  
SMG\$\_INVKBD\_ID                Invalid **keyboard-id**.  
SMG\$\_WRONUMARG                Wrong number of arguments.  
LIB\$\_INSVIRMEM                Insufficient virtual memory.

## Example

```

!+
! This FORTRAN example uses the routine
! SMG$REPLACE_INPUT_LINE to concatenate
! the last 2 lines in the recall buffer.
!-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'
 INCLUDE '($SMGMSG)'
 CHARACTER*20 TEXT, TEXT1

 WRITE (5,*) 'Enter number of lines to save.'
 READ (5,*) R

 S = SMG$CREATE_PASTEBOARD(PBID)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$CREATE_VIRTUAL_DISPLAY(22,70,DID,SMG$M_BORDER)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$CREATE_VIRTUAL_KEYBOARD(KBID,,,R)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$PUT_LINE(DID,'Enter lines of text:')
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$PASTE_VIRTUAL_DISPLAY(DID,PBID,2,2)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))

!+
! Read in lines of text.
!-
 DO 10 I = 1,R
 S = SMG$READ_COMPOSED_LINE(KBID,,TEXT,'Example>',,DID)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 10 CONTINUE

!+
! Recall last 2 lines in the buffer.
!-

 S = SMG$RETURN_INPUT_LINE(KBID,TEXT,,1,LEN)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$RETURN_INPUT_LINE(KBID,TEXT1,,2,LEN1)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 TEXT(LEN:20) = TEXT1(1:LEN1)

!+
! Concatenate them and replace the last 2
! lines in the buffer with the new line.
!-

 S = SMG$REPLACE_INPUT_LINE(KBID,TEXT,2)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))

!+
! Recall the last line which is now the
! concatenated line.
!-

 S = SMG$RETURN_INPUT_LINE(KBID,TEXT,,1)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))

 S = SMG$PUT_LINE(DID,'**** The last line of text is:')
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$PUT_LINE(DID,TEXT)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))

 END

```

One sample of the output generated by this FORTRAN program is as follows:

## SMG\$REPLACE\_INPUT\_LINE

```
$ RUN REPLACE
Enter number of lines to save.
 3
Enter lines of text:
Example> PASTEBOARD
Example> DISPLAY
Example> KEYBOARD
****The last line of text is:
KEYBOARDDISPLAY
```



---

## SMG\$RESTORE\_PHYSICAL\_SCREEN—Restore Physical Pasteboard

The Restore Physical Pasteboard routine rewrites the pasteboard image as it was at the time the SMG\$SAVE\_PHYSICAL\_SCREEN routine was called.

### Format

SMG\$RESTORE\_PHYSICAL\_SCREEN pasteboard-id ,display-id

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### pasteboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard to be restored. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by the SMG\$CREATE\_PASTEBOARD routine.

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display created by the SMG\$SAVE\_PHYSICAL\_SCREEN routine. The **display-id** argument is the address of an unsigned longword that contains this display identifier.

### Description

SMG\$RESTORE\_PHYSICAL\_SCREEN reproduces the pasteboard image saved by the SMG\$SAVE\_PHYSICAL\_SCREEN routine. You must pass the **display-id** returned by the SMG\$SAVE\_PHYSICAL\_SCREEN routine to the SMG\$RESTORE\_PHYSICAL\_SCREEN routine. Note that when performing multiple calls to SMG\$SAVE\_PHYSICAL\_SCREEN and SMG\$RESTORE\_PHYSICAL\_SCREEN, the calls must be performed in a nested fashion; that is, the last pasteboard saved must be the first one restored.

## SMG\$RESTORE\_PHYSICAL\_SCREEN

### Condition Values Returned

SS\$\_NORMAL

Normal successful completion.

SMG\$\_INVDIS\_ID

Invalid **display-id**.

SMG\$\_INVPAS\_ID

Invalid **pasteboard-id**.

---

## SMG\$RETURN\_CURSOR\_POS—Return Cursor Position

The Return Cursor Position routine returns the current virtual cursor position in a specified virtual display.

### Format

SMG\$RETURN\_CURSOR\_POS display-id ,start-row ,start-column

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display whose current virtual cursor position you are requesting. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |

Receives the virtual cursor's current row position within the specified virtual display. The **start-row** argument is the address of a signed longword into which is written the current row position.

## SMG\$RETURN\_CURSOR\_POS

### **start-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | write only        |
| mechanism     | by reference      |

Receives the virtual cursor's current column position within the specified virtual display. The **start-column** argument is the address of a signed longword into which is written the current column position.

### **Description**

SMG\$RETURN\_CURSOR\_POS returns the virtual cursor's current row and column positions in a specified virtual display.

### **Condition Values Returned**

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .   |
| SMG\$_WRONUMARG | Wrong number of arguments.    |

---

## SMG\$RETURN\_INPUT\_LINE—Return Input Line

The Return Input Line routine returns to the caller the requested line from the recall buffer. This line is retrieved either by matching it with a specified string or by specifying the appropriate line number.

### Format

```
SMG$RETURN_INPUT_LINE keyboard-id ,resultant-string [,match-string] [,byte-integer-line-number]
 [,resultant-length] [,character-set]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### keyboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Keyboard identifier. The **keyboard-id** argument is the address of an unsigned longword containing the identifier of the virtual keyboard from which to read.

The virtual keyboard is created by calling the SMG\$CREATE\_VIRTUAL\_KEYBOARD routine.

#### resultant-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

String into which is written the complete recalled line. The **resultant-string** argument is the address of a descriptor pointing to this string.

#### match-string

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Match string to be used when searching for the line to be recalled. The optional **match-string** argument is the address of a descriptor pointing to this match string. The search begins with the last line typed.

#### byte-integer-line-number

|               |                 |
|---------------|-----------------|
| OpenVMS usage | byte_unsigned   |
| type          | byte (unsigned) |
| access        | read only       |
| mechanism     | by reference    |

## SMG\$RETURN\_INPUT\_LINE

Line number to be used when searching for the line to be recalled. The optional **byte-integer-line-number** argument is the address of an unsigned byte containing the number of the line to be recalled. The last line typed is line number 1.

### resultant-length

OpenVMS usage word\_unsigned  
type word (unsigned)  
access write only  
mechanism by reference

Length of the **resultant-string** string. The optional **resultant-length** argument is the address of an unsigned word containing either the number of characters read or the maximum length of **resultant-string**, whichever is less.

### character-set

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access write only  
mechanism by reference

Receives the default character set for the requested line from the recall buffer. The optional **character-set** argument is the address of an unsigned longword that specifies the character set. If the character set is undefined, receives SMG\$C\_UNKNOWN.

| Value                | Character Set         |
|----------------------|-----------------------|
| SMG\$C_ASCII         | ASCII                 |
| SMG\$C_SPEC_GRAPHICS | DEC Special Graphics  |
| SMG\$C_HANYU         | DEC Hanyu             |
| SMG\$C_UNKNOWN       | Unknown character set |

## Description

SMG\$RETURN\_INPUT\_LINE returns to the caller the specified line in the recall buffer. This routine is intended to aid in the implementation of a DCL-style "RECALL" command.

If you specify the **match-string** argument, SMG\$RETURN\_INPUT\_LINE searches for and returns the line that matches the specified string. If you specify the **byte-integer-line-number** argument, SMG\$RETURN\_INPUT\_LINE returns the line that corresponds to the specified line number. If you specify both **match-string** and **byte-integer-line-number**, SMG\$\_INVARG is returned. If you specify **match-string** and a match is not made, SMG\$\_LINNOTFND is returned.

## Condition Values Returned

SS\$\_NORMAL Normal successful completion.  
SMG\$\_INVARG Invalid argument.  
SMG\$\_INVKBD\_ID Invalid **keyboard-id**.

|                 |                                |
|-----------------|--------------------------------|
| SMG\$_LINNOTFND | Matching line was not found.   |
| SMG\$_WRONUMARG | Wrong number of arguments.     |
| LIB\$_xxx       | Any error from LIB\$COPY_R_DX. |

## Example

```

!+
! This FORTRAN example uses the routine
! SMG$RETURN_INPUT_LINE to implement a
! RECALL/ALL command.
!-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'
 INCLUDE '($SMGMSG)'
 CHARACTER*20 TEXT

 WRITE (5,*) 'Enter number of lines to save.'
 READ (5,*) R

 S = SMG$CREATE_PASTEBOARD(PBID)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$CREATE_VIRTUAL_DISPLAY(22,70,DID,SMG$_M_BORDER)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$CREATE_VIRTUAL_KEYBOARD(KBID,,,R)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$PUT_LINE(DID,'Enter lines of text:')
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$PASTE_VIRTUAL_DISPLAY(DID,PBID,2,2)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))

!+
! Read in lines of text.
!-
 DO 10 I = 1,R
 S = SMG$READ_COMPOSED_LINE(KBID,,TEXT,'Example>',,DID)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
10 CONTINUE

 S = SMG$PUT_LINE(DID,'**** The lines of text are:')
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))

!+
! Recall all lines in the buffer.
!-
 DO 30 N = 1,R
 S = SMG$RETURN_INPUT_LINE(KBID,TEXT,,N)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$PUT_LINE(DID,TEXT)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
30 CONTINUE

!+
! Recall the line containing 'fox'
!-
 S = SMG$PUT_LINE(DID,'**** The line containing "fox" is:',2)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))
 S = SMG$RETURN_INPUT_LINE(KBID,TEXT,'FOX')
 IF (S.EQ. SMG$_LINNOTFND) TEXT = 'None found!'
 S = SMG$PUT_LINE(DID,TEXT)
 IF (.NOT. S) CALL LIB$STOP(%VAL(S))

 END

```

One sample of the output generated by this FORTRAN program is as follows:

## SMG\$RETURN\_INPUT\_LINE

```
$ RUN RETURN
Enter number of lines to save.
 3
Enter lines of text:
Example> PASTEBOARD
Example> DISPLAY
Example> KEYBOARD
****The lines of text are:
KEYBOARD
DISPLAY
PASTEBOARD
****The line containing "fox" is:

None found!
```



---

## SMG\$RING\_BELL—Ring the Terminal Bell or Buzzer

The Ring the Terminal Bell or Buzzer routine sounds the terminal bell or buzzer.

### Format

SMG\$RING\_BELL display-id [,number-of-times]

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display for which the bell or buzzer sounds. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### number-of-times

|               |                           |
|---------------|---------------------------|
| OpenVMS usage | longword_signed           |
| type          | longword integer (signed) |
| access        | read only                 |
| mechanism     | by reference              |

Specifies the number of times the bell or buzzer is sounded. The **number-of-times** argument is the address of a signed longword integer that contains the number of times the bell or buzzer is sounded. If **number-of-times** is omitted, 1 is used.

### Description

SMG\$RING\_BELL sounds the bell or buzzer on each pasteboard to which the specified virtual display is pasted. The bell or buzzer sounds the number of times specified; the default number of times is 1.

### Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .   |

Any condition values returned by \$QIOW, LIB\$GET\_VM, LIB\$FREE\_VM.

## SMG\$SAVE\_PHYSICAL\_SCREEN

---

### SMG\$SAVE\_PHYSICAL\_SCREEN—Save Physical Screen

The Save Physical Screen routine saves the contents of the pasteboard so that a later call to SMG\$RESTORE\_PHYSICAL\_SCREEN can restore it.

#### Format

SMG\$SAVE\_PHYSICAL\_SCREEN pasteboard-id ,display-id [,desired-start-row] [,desired-end-row]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### pasteboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard whose contents are to be saved. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by reference        |

Receives the display identifier of the display created to contain the contents of the specified pasteboard. The **display-id** argument is the address of an unsigned longword into which the display identifier is written.

**Display-id** must be passed to the SMG\$RESTORE\_PHYSICAL\_SCREEN routine to restore the saved information.

##### desired-start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the first row to be saved. The **desired-start-row** argument is the address of a signed longword that contains the row number. If **desired-start-row** is omitted, row 1 of the pasteboard is used.

##### desired-end-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the last row to be saved. The **desired-end-row** argument is the address of a signed longword that contains the row number. If **desired-end-row** is omitted, the last row of the pasteboard is used.

## Description

SMG\$SAVE\_PHYSICAL\_SCREEN blanks the screen by creating a virtual display that is as wide as the specified pasteboard and as high as specified by the **desired-start-row** and **desired-end-row** arguments. If these two arguments are omitted, the created virtual display is as high as the specified pasteboard.

The created virtual display does not have a default character-set, so that display output to the virtual display should consider its character-set.

The information saved — that is, the pasteboard image — can be restored by calling the SMG\$RESTORE\_PHYSICAL\_SCREEN routine. When performing multiple calls to SMG\$SAVE\_PHYSICAL\_SCREEN and SMG\$RESTORE\_PHYSICAL\_SCREEN, the calls must be performed in a nested order; that is, the last pasteboard saved must be the first one restored, and so on.

These routines are useful when calling a procedure that may send output to the screen without using the Screen Management Facility. Before calling such a procedure, you save the pasteboard image with SMG\$SAVE\_PHYSICAL\_SCREEN. After the procedure executes, you restore the pasteboard image with SMG\$RESTORE\_PHYSICAL\_SCREEN.

Note that the saved region must encompass at least two rows. When you use SMG\$SAVE\_PHYSICAL\_SCREEN on a terminal that does not support scrolling regions, you must save and restore the entire pasteboard.

## Condition Values Returned

|                 |                                                                       |
|-----------------|-----------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                         |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> .                                        |
| SMG\$_INVROW    | Invalid row, or range specified does not encompass at least two rows. |
| SMG\$_WRONUMARG | Wrong number of arguments.                                            |
| LIB\$_INSVIRMEM | Insufficient virtual memory.                                          |

## SMG\$SAVE\_VIRTUAL\_DISPLAY

---

### SMG\$SAVE\_VIRTUAL\_DISPLAY—Save the Virtual Display to a File

The Save the Virtual Display to a File routine saves the contents of a virtual display and stores it in a file.

#### Format

SMG\$SAVE\_VIRTUAL\_DISPLAY display-id [,filespec]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display to be saved. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### filespec

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

String containing the file specification of the file in which the specified virtual display is saved. The **filespec** argument is the address of a descriptor pointing to the character string containing the file specification.

A new file is created each time this routine is called. If **filespec** is omitted, the default file specification is SMGDISPLY.DAT.

#### Description

SMG\$SAVE\_VIRTUAL\_DISPLAY saves the contents of a virtual display and stores it in a nonprintable file. The text, renditions, character sets, and all attributes necessary to reconstruct the virtual display are saved. Menu, viewport, and subprocess context are not saved. The SMG\$LOAD\_VIRTUAL\_DISPLAY routine restores the virtual display.

**Condition Values Returned**

SS\$\_NORMAL

Normal successful completion.

SMG\$\_INVDIS\_ID

Invalid **display-id**.

RMS\$\_xxxx

Any error returned by \$OPEN, \$CONNECT,  
\$PUT, \$CLOSE.

## SMG\$SCROLL\_DISPLAY\_AREA

---

### SMG\$SCROLL\_DISPLAY\_AREA—Scroll Display Area

The Scroll Display Area routine scrolls a rectangular region of a virtual display.

#### Format

```
SMG$SCROLL_DISPLAY_AREA display-id [,start-row] [,start-column] [,height] [,width] [,direction]
 [,count]
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **display-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display in which scrolling takes place. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### **start-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the first row of the scrolling region. The **start-row** argument is the address of a signed longword that contains the starting row.

If **start-row** is omitted, row 1 of the specified virtual display is used. Note that if you omit either **start-row** or **start-column**, the default (row 1 and column 1) is used.

##### **start-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the first column of the scrolling region. The **start-column** argument is the address of a signed longword that contains the starting column.

If omitted, column 1 of the specified virtual display is used. Note that if you omit either **start-row** or **start-column**, the default (row 1 and column 1) is used.

In the case that the **start-column** begins on the right portion of a Chinese character, the character will become two undefined characters.

### **height**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the number of rows in the scrolling region. The **height** argument is the address of a signed longword that contains the number of rows.

If omitted, this value defaults to either the height of the virtual scrolling region (if one has been explicitly set with SMG\$SET\_DISPLAY\_SCROLL\_REGION) or the height of the specified virtual display.

When scrolling a portion of a virtual display that contains highwide characters, it is recommended that you scroll the display an even number of times.

### **width**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the number of columns in the scrolling region. The **width** argument is the address of a signed longword that contains the number of columns.

If omitted, this value defaults to the width of the specified virtual display.

## SMG\$SCROLL\_DISPLAY\_AREA

### direction

OpenVMS usage longword\_unsigned  
type longword (unsigned)  
access read only  
mechanism by reference

Specifies the direction to scroll. The **direction** argument is the address of an unsigned longword that contains the direction code.

Valid values are SMG\$M\_UP, SMG\$M\_DOWN, SMG\$M\_RIGHT, and SMG\$M\_LEFT. SMG\$M\_UP is the default.

### count

OpenVMS usage longword\_signed  
type longword (signed)  
access read only  
mechanism by reference

Specifies the number of lines or columns to scroll. The **count** argument is the address of a signed longword that contains the number of units (lines or columns) to scroll. If omitted, one unit is scrolled.

## Description

SMG\$SCROLL\_DISPLAY\_AREA scrolls a rectangular region of the specified virtual display. It scrolls the region a specified number of lines or columns in the specified direction. After the scroll, the cursor is left at the **start-row**, **start-column** position.

In the case that the Chinese language character set is used, if the **start-column** begins on the right portion of the Chinese character, the Chinese character will become two undefined characters. If the right most column of the rectangle region contains the left portion of the Chinese character, the Chinese character will become two undefined characters.

Also, if the rectangle region scrolls to the left and the left most column of the scrolled rectangle region contains the right portion of the Chinese character, the right portion of the Chinese character will become an undefined character. If the rectangle region scrolls to the right and the right most column of the scrolled rectangle region contains the left portion of the Chinese character, the left portion of the Chinese character will become an undefined character.

## Condition Values Returned

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                         |
| SMG\$_INVARG    | The defined rectangle is outside the virtual display. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                           |
| SMG\$_INVCOL    | Invalid column.                                       |
| SMG\$_INVROW    | Invalid row.                                          |
| SMG\$_WRONUMARG | Wrong number of arguments.                            |



---

## SMG\$SCROLL\_VIEWPORT—Scroll a Display Under a Viewport

The Scroll a Display Under a Viewport routine scrolls a virtual display under its associated viewport.

### Format

```
SMG$SCROLL_VIEWPORT display-id [,direction] [,count]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Display identifier of the virtual display to be scrolled. The **display-id** argument is the address of an unsigned longword containing this identifier.

#### direction

|               |                     |
|---------------|---------------------|
| OpenVMS usage | longword_unsigned   |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional direction specifier. The **direction** argument is the address of an unsigned longword that contains the direction code specifying the scrolling direction. The optional **count** argument can be used to specify the number of lines or columns to scroll in the specified direction. Valid values for **direction** are as follows:

|              |                                           |
|--------------|-------------------------------------------|
| SMG\$M_UP    | Scroll <b>count</b> lines upward.         |
| SMG\$M_DOWN  | Scroll <b>count</b> lines downward.       |
| SMG\$M_RIGHT | Scroll <b>count</b> columns to the right. |
| SMG\$M_LEFT  | Scroll <b>count</b> columns to the left.  |

SMG\$M\_UP is the default.

#### count

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Optional number of rows or columns that will be scrolled. The default is 1.

## SMG\$SCROLL\_VIEWPORT

### Description

SMG\$SCROLL\_VIEWPORT scrolls a virtual display under its associated viewport. The viewport is actually changing its coordinates as it moves over the virtual display to simulate scrolling; however, it does not change its physical location on the screen. The size of the viewport could change if the viewport moves off the virtual display. To restore the size of the viewport, use SMG\$CHANGE\_VIEWPORT.

In case Chinese language character set is used, and the left most column of the viewport contains the right portion of a Chinese character, the right portion of the character becomes invisible. Also if the right most column of the viewport contains the left portion of a Chinese character, the left portion of the character becomes invisible.

### Condition Values Returned

|                   |                                                                                                             |
|-------------------|-------------------------------------------------------------------------------------------------------------|
| SS\$_NORMAL       | Normal successful completion.                                                                               |
| SMG\$_INVARG      | The value of <b>count</b> is less than zero.                                                                |
| SMG\$_INVDIS_ID   | Invalid <b>display-id</b> .                                                                                 |
| SMG\$_NO_WINASSOC | No viewport is associated with the specified virtual display.                                               |
| SMG\$_WINTRUFIT   | Successful completion; however, the viewport associated with the virtual display has been truncated to fit. |
| SMG\$_WRONUMARG   | Wrong number of arguments.                                                                                  |

### Example

```
C This DEC Fortran example demonstrates SMG$SCROLL_VIEWPORT.
C Include the SMG definitions. In particular, we want SMG$M_BORDER.

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'

C Create the Virtual Displays. Give them borders.

 ROWS = 10
 COLUMNS = 22

 STATUS = SMG$CREATE_VIRTUAL_DISPLAY
1 (ROWS, COLUMNS, DISPLAY1, SMG$M_BORDER)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C Create viewport

 STATUS = SMG$CREATE_VIEWPORT (DISPLAY1, 2, 2, 3, 10)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C Create the Pasteboard

 STATUS = SMG$CREATE_PASTEBOARD (PASTE1)
 IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

C Put data into the Virtual Displays
```

## SMG\$SCROLL\_VIEWPORT

```
STATUS = SMG$PUT_CHARS (DISPLAY1, '11111111111111', 1, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
STATUS = SMG$PUT_CHARS (DISPLAY1, 'This is row 2.', 2, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
STATUS = SMG$PUT_CHARS (DISPLAY1, '33333333333333', 3, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
STATUS = SMG$PUT_CHARS (DISPLAY1, 'This is row 4.', 4, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = SMG$PUT_CHARS (DISPLAY1, '55555555555555', 5, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
STATUS = SMG$PUT_CHARS (DISPLAY1, 'This is row 6.', 6, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
STATUS = SMG$PUT_CHARS (DISPLAY1, '77777777777777', 7, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
STATUS = SMG$PUT_CHARS (DISPLAY1, 'This is row 8.', 8, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
STATUS = SMG$PUT_CHARS (DISPLAY1, '99999999999999', 9, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
STATUS = SMG$PUT_CHARS (DISPLAY1, 'This is row 10.', 10, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

C Paste the Virtual Display

```
STATUS = SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PASTE1, 4, 3)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))

STATUS = LIB$WAIT (2.0)

DO 1 I = 1, 3
STATUS = SMG$SCROLL_VIEWPORT(DISPLAY1, SMG$M_UP, 1)
IF (.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
CALL LIB$WAIT (2.0)
1 CONTINUE

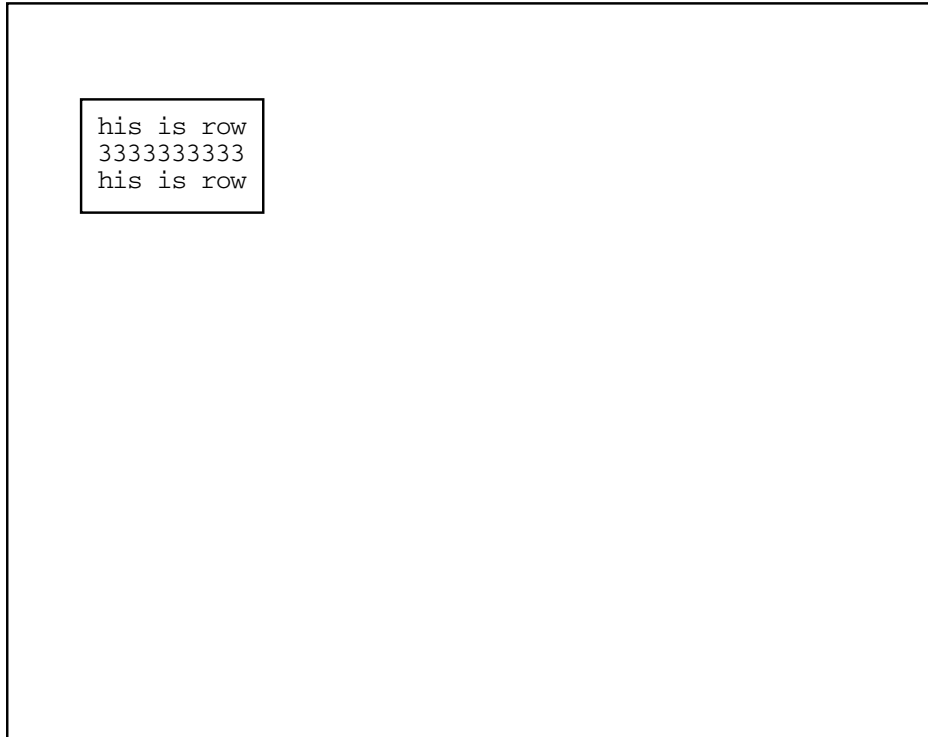
STATUS = LIB$WAIT (2.0)

END
```

In the preceding example, a single virtual display is created and associated with a viewport. Since the virtual display is pasted to the pasteboard after it is associated with the viewport, only the portion of the virtual display that falls inside the viewport is visible. This is displayed in Figure SMG-45.

## SMG\$SCROLL\_VIEWPORT

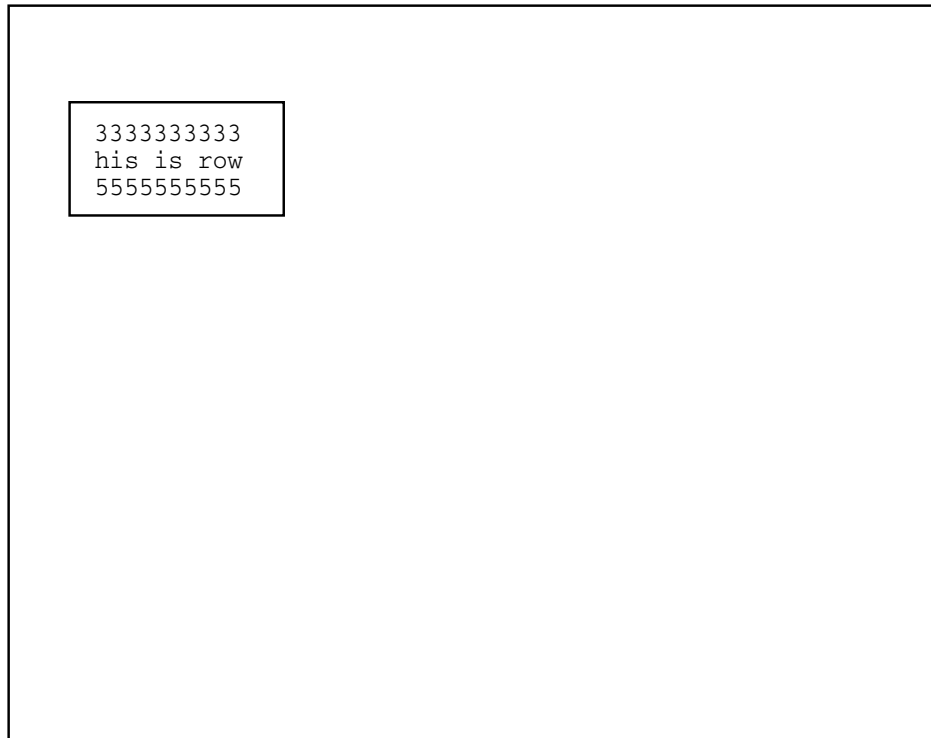
Figure SMG-45 Output Generated by Pasting the Virtual Display



ZK-6426/1-GE

The call to SMG\$SCROLL\_VIEWPORT is repeated a total of three times. Figure SMG-46 shows the viewport after the first call to SMG\$SCROLL\_VIEWPORT.

Figure SMG-46 Output Generated After First Call to SMG\$SCROLL\_VIEWPORT

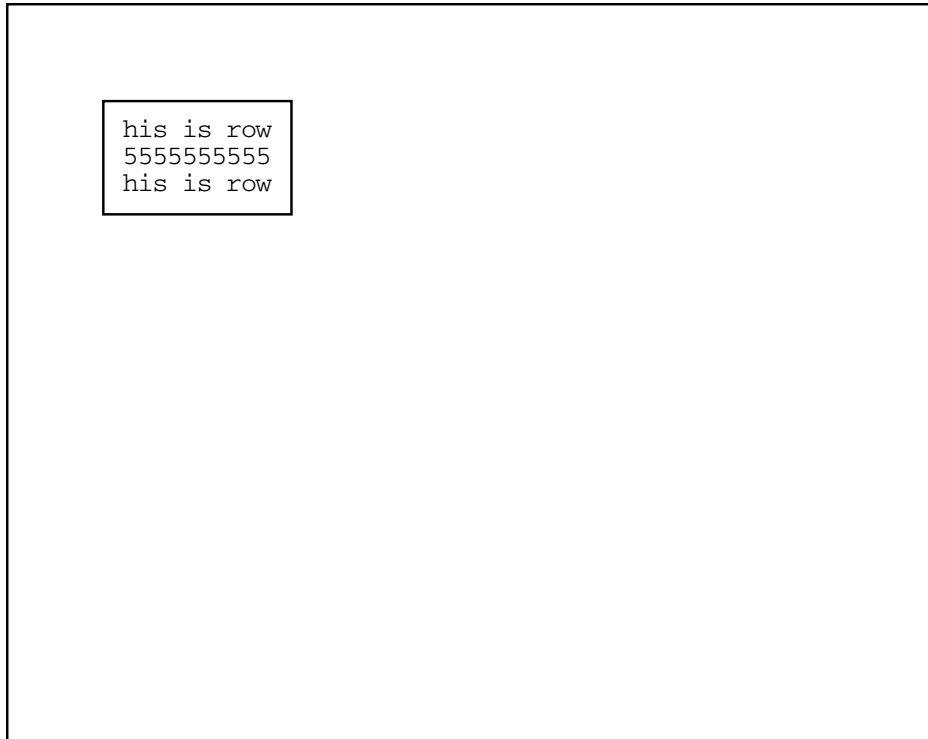


ZK-6426/2-GE

Figure SMG-47 shows the contents of the viewport after the second call to SMG\$SCROLL\_VIEWPORT.

## SMG\$SCROLL\_VIEWPORT

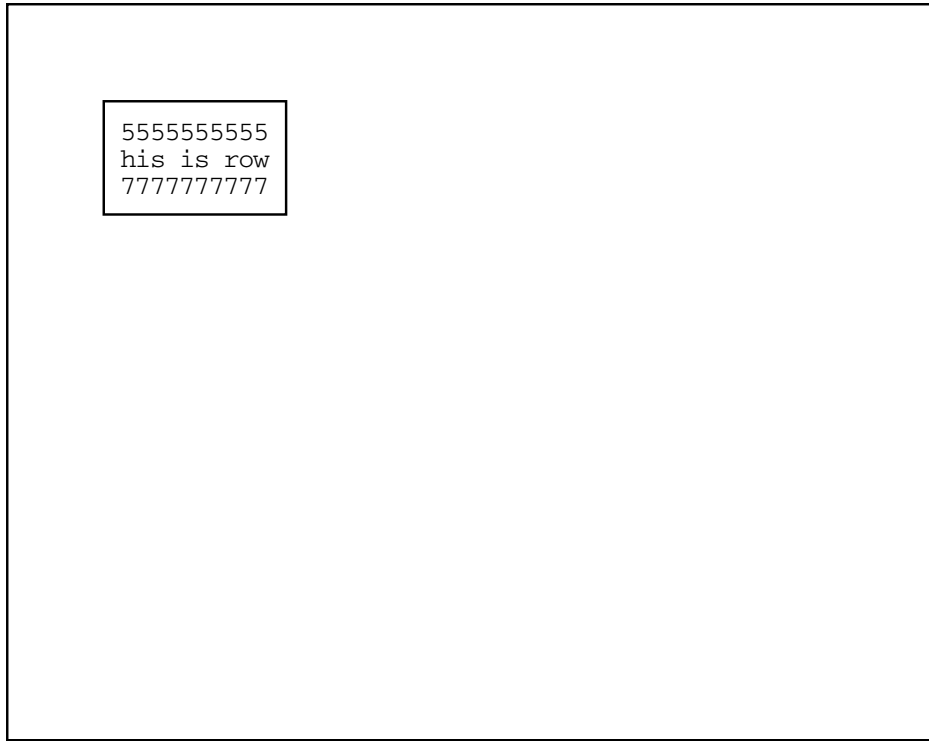
Figure SMG-47 Output Generated After the Second Call to SMG\$SCROLL\_VIEWPORT



ZK-6426/3-GE

Figure SMG-48 shows the contents of the viewport after the last call to SMG\$SCROLL\_VIEWPORT.

Figure SMG-48 Output Generated After the Last Call to SMG\$SCROLL\_VIEWPORT



ZK-6426/4-GE

## SMG\$SELECT\_FROM\_MENU

---

### SMG\$SELECT\_FROM\_MENU—Make a Selection from the Menu

The Make a Selection from the Menu routine lets you move between the menu choices using the arrow keys and lets you make a selection by pressing the Return key.

#### Format

```
SMG$SELECT_FROM_MENU keyboard-id ,display-id ,selected-choice-number [,default-choice-number]
 [,flags] [,help-library] [,timeout] [,word-terminator-code]
 [,selected-choice-string] [,rendition-set] [,rendition-complement]
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### keyboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the virtual keyboard from which the terminal user's responses are read. The **keyboard-id** argument is the address of an unsigned longword containing this identifier.

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Identifier of the virtual display in which the choices are displayed. The **display-id** argument is the address of an unsigned longword containing this display identifier. This virtual display must be pasted to a pasteboard and cannot be batched or occluded.

##### selected-choice-number

|               |                 |
|---------------|-----------------|
| OpenVMS usage | word_unsigned   |
| type          | word (unsigned) |
| access        | write only      |
| mechanism     | by reference    |

Identification number of the menu item selected. The **selected-choice-number** argument is the address of an unsigned word that receives this number. The **selected-choice-number** corresponds to the index of the menu item in the static string array specified in SMG\$CREATE\_MENU.



**default-choice-number**

OpenVMS usage word\_unsigned  
 type word (unsigned)  
 access read only  
 mechanism by reference

Optional identification number of the default menu item. The **default-choice-number** argument is the address of an unsigned word that contains the number of the default menu item. The **default-choice-number** corresponds to the index of the default menu item in the static string array specified in SMG\$CREATE\_MENU. If omitted, the default choice will be the last menu item already selected, or the first item in the menu if no selections have yet been made.

**flags**

OpenVMS usage mask\_longword  
 type longword (unsigned)  
 access read only  
 mechanism by reference

Optional bit mask indicating behavior when a selection is made. The **flags** argument is the address of an unsigned longword containing the flag. Valid values are as follows:

|                     |                                                                                |
|---------------------|--------------------------------------------------------------------------------|
| SMG\$M_RETURN_IMMED | Returns control to the user when any key other than an arrow key is entered.   |
| SMG\$M_REMOVE_ITEM  | Causes SMG\$SELECT_FROM_MENU to allow each menu item to be selected only once. |

**help-library**

OpenVMS usage char\_string  
 type character string  
 access read only  
 mechanism by descriptor

Optional help library to use if the PF2/HELP key is pressed. The **help-library** argument is the address of a descriptor pointing to this help library name. Note that this argument is ignored if **flags** specifies SMG\$M\_RETURN\_IMMED. The default is SYS\$HELP:HELPLIB.HLB.

If the menu item does not consist of ASCII characters, the menu item is not used to get the help text but just invokes help facility to specify the help topic to get some information.

**timeout**

OpenVMS usage longword\_signed  
 type longword (signed)  
 access read only  
 mechanism by reference

Optional timeout value. The **timeout** argument is the address of a signed longword that specifies the number of seconds to wait for a selection to be made.

**word-terminator-code**

OpenVMS usage word\_unsigned  
 type word (unsigned)  
 access write only  
 mechanism by reference

## SMG\$SELECT\_FROM\_MENU

Optional unsigned word that receives the code indicating which key terminated the read. The **word-terminator-code** argument is the address of an unsigned word that receives this terminating key code.

### selected-choice-string

OpenVMS usage char\_string  
type character string  
access write only  
mechanism by descriptor

Optional string that receives the text of the menu item selected. The **selected-choice-string** is the address of a descriptor pointing to this string.

### rendition-set

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Attribute specifier. The optional **rendition-set** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be set in the display. The following attributes can be specified using the **rendition-set** argument:

|                                         |                                                                                                                    |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| SMG\$M_BLINK                            | Displays blinking characters.                                                                                      |
| SMG\$M_BOLD                             | Displays characters in higher-than-normal intensity.                                                               |
| SMG\$M_REVERSE                          | Displays characters in reverse video; that is, using the opposite of the default rendition of the virtual display. |
| SMG\$M_UNDERLINE                        | Displays underlined characters.                                                                                    |
| SMG\$M_USER1<br>through<br>SMG\$M_USER8 | Displays user-defined attributes.                                                                                  |

The **display-id** argument must be specified when you use the **rendition-set** argument.

### rendition-complement

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Attribute complement specifier. The optional **rendition-complement** argument is the address of a longword bit mask in which each attribute set causes the corresponding attribute to be complemented in the display. All attributes that can be specified with the **rendition-set** argument can be complemented with the **rendition-complement** argument. The **display-id** argument must be specified when you use the **rendition-complement** argument.

The optional arguments **rendition-set** and **rendition-complement** let the user control the attributes of the virtual display. The **rendition-set** argument sets certain virtual display attributes, while **rendition-complement** complements these attributes. If the same bit is specified in both the **rendition-set** and **rendition-complement** parameters, **rendition-set** is evaluated first, followed by **rendition-complement**. By using these two parameters together, the user can control each virtual display attribute in a single procedure call. On a single-attribute basis, the user can cause the following transformations:

| Set | Complement | Action                                         |
|-----|------------|------------------------------------------------|
| 0   | 0          | Attribute set to default                       |
| 1   | 0          | Attribute on                                   |
| 0   | 1          | Attribute set to complement of default setting |
| 1   | 1          | Attribute off                                  |

## Description

SMG\$SELECT\_FROM\_MENU lets you make a selection from the items in the menu. Note that the routine SMG\$CREATE\_MENU must be called before calling SMG\$SELECT\_FROM\_MENU.

You can move between the various menu items using the arrow keys, and you make a selection by pressing the Return key. Typing a Ctrl/Z selects the current choice and returns the value SMG\$\_EOF. Note that if there are more menu choices than can be displayed, the display is scrolled, the appropriate arrow key is typed, and the additional menu choices become visible.

The current selection is indicated in reverse video (or in the rendition specified) and by the physical cursor. The selected choice is returned to the caller in the **selected-choice-number** argument. In addition, the selected item will be removed from the remaining menu items if SMG\$M\_REMOVE\_ITEM is specified for the **flags** parameter.

SMG\$SELECT\_FROM\_MENU has two modes of operation that are specified using the **flags** argument. A value without SMG\$M\_RETURN\_IMMED (the default) causes the following behavior:

- The user can move among the choices using the arrow keys.
- The only keys that select the current item are the following:
  - Return
  - DO
  - SELECT
  - Enter
- HELP or PF2 outputs help for the current item.
- Ctrl/W refreshes the screen by calling SMG\$REPAINT\_SCREEN.
- Ctrl/Z selects the current item and returns a value of SMG\$\_EOF.
- PF1/UP\_ARROW selects the first item in the menu.
- PF1/DOWN\_ARROW selects the last item in the menu.
- PF1/LEFT\_ARROW selects the first item in the current row.
- PF1/RIGHT\_ARROW selects the last item in the current row.
- All other keys are ignored.

A value of SMG\$M\_RETURN\_IMMED enables the following:

- The user can move among the menu choices using the arrow keys.
- Ctrl/Z selects the current item and returns SMG\$\_EOF.
- Any other key entered selects the current item.

## SMG\$SELECT\_FROM\_MENU

In either case, the following key substitutions can be made:

- The LF key can be used in place of Down Arrow.
- The BACKSPACE key can be used in place of Left Arrow.
- The TAB key can be used in place of Right Arrow.

The SMG\$K\_TRM\_keyname code for the terminating key is returned in the optional **word-terminator-code** argument. Multiword menu items are allowed.

When the menu items are not ASCII characters, the user can not display the help text for the menu items by the PF2/HELP key, but just invoke the help facility.

### Condition Values Returned

|                 |                                                                                                                                                                                                                                        |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                                                                                                                                                                                          |
| SMG\$_EOF       | End of file.                                                                                                                                                                                                                           |
| SMG\$_NOTPASTED | The virtual display is not pasted.                                                                                                                                                                                                     |
| SMG\$_INVDIS_ID | The <b>display-id</b> is invalid, does not contain a menu, or contains a viewport.                                                                                                                                                     |
| SMG\$_ILLBATFNC | The virtual display or pasteboard is batched.                                                                                                                                                                                          |
| SMG\$_INVARG    | Invalid argument or none of the menu items is selectable.                                                                                                                                                                              |
| LIB\$_xxxx      | Any completion status returned by LIB\$SCOPY.                                                                                                                                                                                          |
| SMG\$_xxxx      | Any completion status returned by SMG\$CHANGE_RENDITION, SMG\$SET_CURSOR_ABS, SMG\$BEGIN_DISPLAY_UPDATE, SMG\$END_DISPLAY_UPDATE, SMG\$READ_KEYSTROKE, SMG\$PUT_HELP_TEXT, SMG\$SAVE_PHYSICAL_SCREEN, or SMG\$RESTORE_PHYSICAL_SCREEN. |

### Examples

1. 

```
! +
! This VAX Pascal program demonstrates the use of SMG$CREATE_MENU and
! SMG$SELECT_FROM_MENU. This program creates a block menu
! and allows the user to make selections from the menu.
! -

[INHERIT ('SYS$LIBRARY:STARLET')]
PROGRAM BLOCK_MENU (INPUT,OUTPUT);

CONST
 NULL = 0;

TYPE
 CHAR_STRING = VARYING [20] OF CHAR;
 WORD = [WORD] 0..65535;
 FIXED_STRING = PACKED ARRAY[1..9] OF CHAR;

VAR
 OPTIONS : ARRAY[1..9] OF FIXED_STRING;
 I : INTEGER;
 RET_STATUS : UNSIGNED;
 SELECTED : FIXED_STRING;
 NUMBER, DEF_NUMBER : WORD;
 PB_ID, KB_ID, DISPLAY1, DISPLAY2 : UNSIGNED;
 TERM : WORD;
```

## SMG\$SELECT\_FROM\_MENU

```
[EXTERNAL] FUNCTION SMG$CREATE_PASTEBOARD(
 VAR PASTEBOARD_ID : UNSIGNED
) : INTEGER; EXTERN;

[EXTERNAL] FUNCTION SMG$CREATE_VIRTUAL_KEYBOARD(
 VAR KEYBOARD_ID : UNSIGNED
) : INTEGER; EXTERN;

[EXTERNAL] FUNCTION SMG$CREATE_VIRTUAL_DISPLAY(
 NUM_ROWS : INTEGER;
 NUM_COLS : INTEGER;
 VAR DISPLAY_ID : UNSIGNED;
 ATTRIBUTES : UNSIGNED
) : INTEGER; EXTERN;

[EXTERNAL] FUNCTION SMG$PASTE_VIRTUAL_DISPLAY(
 DISPLAY_ID : UNSIGNED;
 PASTEBOARD_ID : UNSIGNED;
 ROW : INTEGER;
 COL : INTEGER
) : INTEGER; EXTERN;

[EXTERNAL] FUNCTION SMG$CREATE_MENU(
 DISPLAY_ID : UNSIGNED;
 CHOICES : ARRAY[A..B : INTEGER] OF FIXED_STRING;
 MENU_TYPE : UNSIGNED;
 MENU_FLAGS : UNSIGNED;
 ROW : INTEGER := %IMMED 0;
 REND_SET : UNSIGNED := %IMMED 0;
 REND_COMP : UNSIGNED
) : INTEGER; EXTERN;

[EXTERNAL] FUNCTION SMG$SELECT_FROM_MENU(
 KEYBOARD_ID : UNSIGNED;
 DISPLAY_ID : UNSIGNED;
 VAR SELEC_NUM : WORD;
 DEFAULT_NUM : WORD;
 MENU_FLAGS : UNSIGNED;
 HELP_LIBR : CHAR_STRING := %IMMED 0;
 TIMEOUT : INTEGER := %IMMED 0;
 VAR TERM_CODE : WORD;
 VAR SELEC_STR : FIXED_STRING
) : INTEGER; EXTERN;

[EXTERNAL] FUNCTION SMG$PUT_LINE(
 DISPLAY_ID : UNSIGNED;
 TEXT : CHAR_STRING
) : INTEGER; EXTERN;

[EXTERNAL] FUNCTION SMG$DELETE_MENU(
 DISPLAY_ID : UNSIGNED
) : INTEGER; EXTERN;

[EXTERNAL] FUNCTION LIB$STOP(
 CONDITION_STATUS : [IMMEDIATE,UNSAFE] UNSIGNED
) : INTEGER; EXTERN;

BEGIN
DEF_NUMBER := 5;
```

## SMG\$SELECT\_FROM\_MENU

```
OPTIONS[1] := 'Northwest';
OPTIONS[2] := 'North ';
OPTIONS[3] := 'Northeast';
OPTIONS[4] := 'West ';
OPTIONS[5] := 'Equator ';
OPTIONS[6] := 'East ';
OPTIONS[7] := 'Southwest';
OPTIONS[8] := 'South ';
OPTIONS[9] := 'Southeast';

RET_STATUS := SMG$CREATE_PASTEBOARD (PB_ID);
IF NOT ODD(RET_STATUS)
THEN
 LIB$STOP(RET_STATUS);

RET_STATUS := SMG$CREATE_VIRTUAL_KEYBOARD (KB_ID);
IF NOT ODD(RET_STATUS)
THEN
 LIB$STOP(RET_STATUS);

RET_STATUS := SMG$CREATE_VIRTUAL_DISPLAY (3, 12, DISPLAY2, SMG$M_BORDER);
IF NOT ODD(RET_STATUS)
THEN
 LIB$STOP(RET_STATUS);

RET_STATUS := SMG$CREATE_VIRTUAL_DISPLAY (6, 37, DISPLAY1, SMG$M_BORDER);
IF NOT ODD(RET_STATUS)
THEN
 LIB$STOP(RET_STATUS);

RET_STATUS := SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY2, PB_ID, 2, 16);
IF NOT ODD(RET_STATUS)
THEN
 LIB$STOP(RET_STATUS);

RET_STATUS := SMG$PASTE_VIRTUAL_DISPLAY (DISPLAY1, PB_ID, 10, 10);
IF NOT ODD(RET_STATUS)
THEN
 LIB$STOP(RET_STATUS);

RET_STATUS := SMG$CREATE_MENU (DISPLAY1, OPTIONS, SMG$K_BLOCK,
 SMG$M_DOUBLE_SPACE,,, SMG$M_BOLD);
IF NOT ODD(RET_STATUS)
THEN
 LIB$STOP(RET_STATUS);

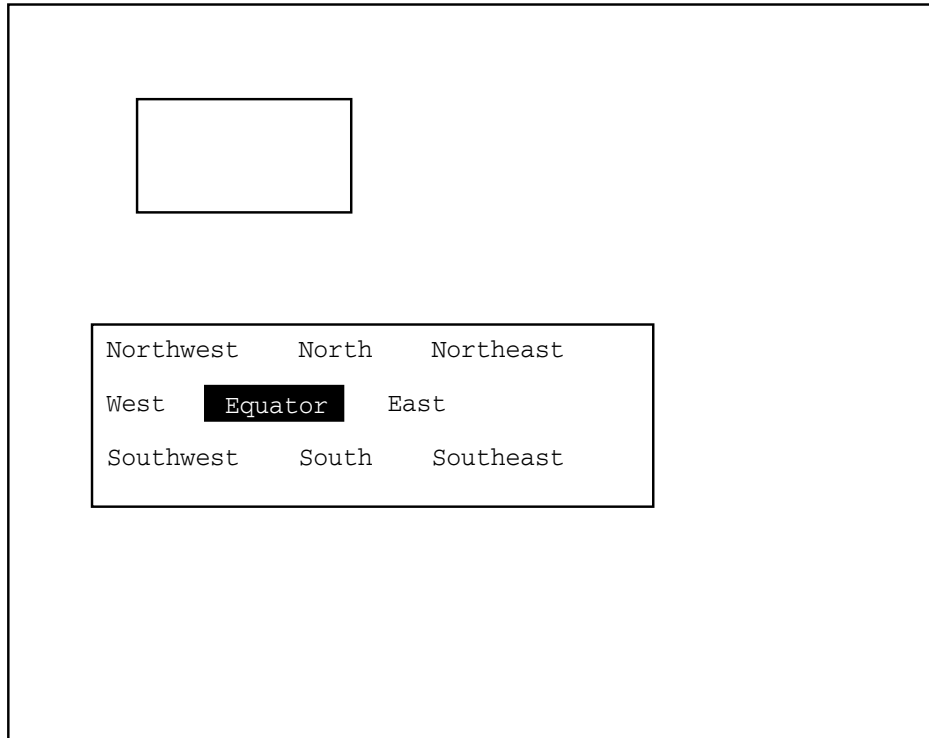
RET_STATUS := SMG$SELECT_FROM_MENU (KB_ID, DISPLAY1, NUMBER, DEF_NUMBER,
 SMG$M_RETURN_IMMED,,, TERM, %DESCR SELECTED);
IF NOT ODD(RET_STATUS)
THEN
 LIB$STOP(RET_STATUS);

RET_STATUS := SMG$PUT_LINE (DISPLAY2, %DESCR SELECTED);
IF NOT ODD(RET_STATUS)
THEN
 LIB$STOP(RET_STATUS);

END.
```

The output for this VAX Pascal program is shown in the following figures. In Figure SMG-49, the program is waiting for the user to make a menu selection.

Figure SMG-49 Output Generated Before a Menu Selection Is Made

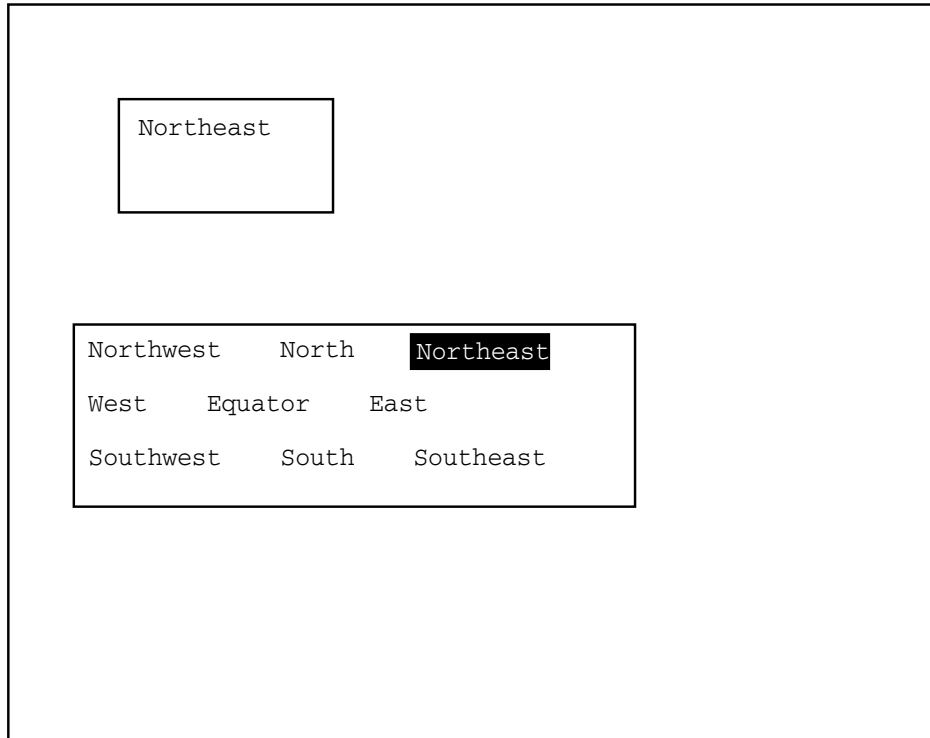


ZK-6427/1-GE

Because the menu is created using the SMG\$M\_RETURN\_IMMED attribute, once the user makes a selection the menu is terminated, and control returns to the program. The menu item selected by the user is displayed in the upper virtual display. This output is shown in Figure SMG-50.

# SMG\$SELECT\_FROM\_MENU

Figure SMG-50 Output Generated After the User Selects an Item



ZK-6427/2-GE



## SMG\$SELECT\_FROM\_MENU

```
2. 10 !+
 !This VAX BASIC program demonstrates the use of
 !SMG-supported menus. Using SMG$CREATE_MENU and
 !SMG$SELECT_FROM_MENU, this program creates an
 !application that uses a vertical menu and allows
 !the user to make multiple selections.
 !-

 OPTION TYPE = EXPLICIT

 EXTERNAL SUB LIB$STOP (LONG BY VALUE)
 EXTERNAL LONG FUNCTION SMG$CREATE_PASTEBOARD (LONG)
 EXTERNAL LONG FUNCTION SMG$CREATE_VIRTUAL_KEYBOARD (LONG)
 EXTERNAL LONG FUNCTION SMG$CREATE_VIRTUAL_DISPLAY (LONG, LONG, &
 LONG, LONG, LONG)
 EXTERNAL LONG FUNCTION SMG$PASTE_VIRTUAL_DISPLAY (LONG, LONG, &
 LONG, LONG)
 EXTERNAL LONG FUNCTION SMG$CREATE_MENU (LONG, STRING DIM(), LONG, &
 LONG, LONG, LONG, LONG)
 EXTERNAL LONG FUNCTION SMG$SELECT_FROM_MENU (LONG, LONG, WORD, &
 WORD, LONG, STRING, LONG, WORD, STRING)
 EXTERNAL LONG FUNCTION SMG$PUT_LINE (LONG, STRING)
 EXTERNAL LONG FUNCTION SMG$DELETE_MENU (LONG)

 %INCLUDE "$SMGDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"

 DECLARE STRING chosen
 MAP (xyz) STRING choice(20) = 16
 DECLARE LONG ret_status, pasteboard_id, display1_id, display2_id, &
 keyboard_id
 DECLARE WORD number

 choice(0) = "ONE"
 choice(1) = "TWO"
 choice(2) = "THREE"
 choice(3) = "FOUR"
 choice(4) = "FIVE"
 choice(5) = "SIX"
 choice(6) = "SEVEN"
 choice(7) = "EIGHT"
 choice(8) = "NINE"
 choice(9) = "TEN"
 choice(10) = "ELEVEN"
 choice(11) = "TWELVE"
 choice(12) = "THIRTEEN"
 choice(13) = "FOURTEEN"
 choice(14) = "FIFTEEN"
 choice(15) = "SIXTEEN"
 choice(16) = "SEVENTEEN"
 choice(17) = "EIGHTEEN"
 choice(18) = "NINETEEN"
 choice(19) = "TWENTY"
 choice(20) = "Exit"

 ret_status = SMG$CREATE_PASTEBOARD (pasteboard_id)
 IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP (ret_status BY VALUE)
 END IF

 ret_status = SMG$CREATE_VIRTUAL_KEYBOARD (keyboard_id)
 IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP (ret_status BY VALUE)
 END IF

 ret_status = SMG$CREATE_VIRTUAL_DISPLAY (10, 20, display1_id, &
 SMGM_BORDER, SMGM_BOLD)
 IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP (ret_status BY VALUE)
 END IF
```

## SMG\$SELECT\_FROM\_MENU

```
ret_status = SMG$CREATE_VIRTUAL_DISPLAY (6, 20, display2_id, &
 SMG$M_BORDER,)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP (ret_status BY VALUE)
END IF

ret_status = SMG$PASTE_VIRTUAL_DISPLAY (display2_id, &
 pasteboard_id, 17, 20)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP (ret_status BY VALUE)
END IF

ret_status = SMG$PASTE_VIRTUAL_DISPLAY (display1_id, &
 pasteboard_id, 4, 20)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP (ret_status BY VALUE)
END IF

ret_status = SMG$CREATE_MENU (display1_id, choice(), &
 SMG$K_VERTICAL,,,SMG$M_BOLD, SMG$M_BOLD)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP (ret_status BY VALUE)
END IF

20 ret_status = SMG$SELECT_FROM_MENU (keyboard_id, display1_id, &
 number,, SMG$M_REMOVE_ITEM,,,,chosen)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP (ret_status BY VALUE)
END IF

ret_status = SMG$PUT_LINE (display2_id, chosen)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP (ret_status BY VALUE)
END IF

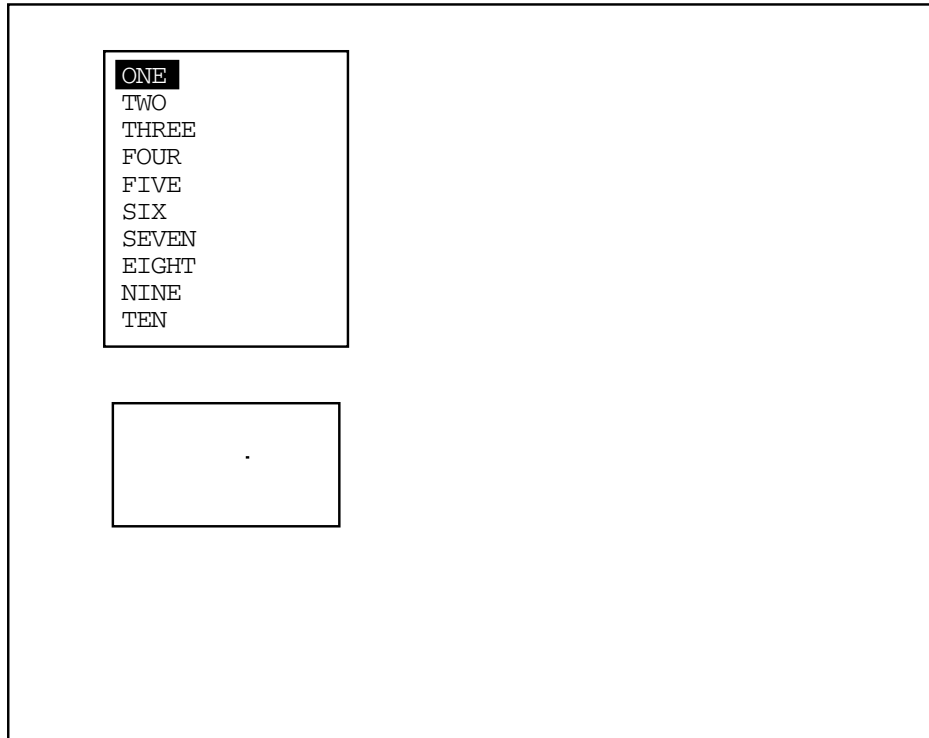
IF (number <> 20) THEN
 GOTO 20
END IF

ret_status = SMG$DELETE_MENU (display1_id)
IF (ret_status AND 1%) = 0% THEN
 CALL LIB$STOP (ret_status BY VALUE)
END IF

END
```

The vertical menu generated by this VAX BASIC program is shown in the following figures. The default choice is set to the first item in the menu: "ONE". In Figure SMG-51, the program is waiting for the user to make a selection from the menu.

Figure SMG-51 Output Generated Before the User Selects a Menu Item

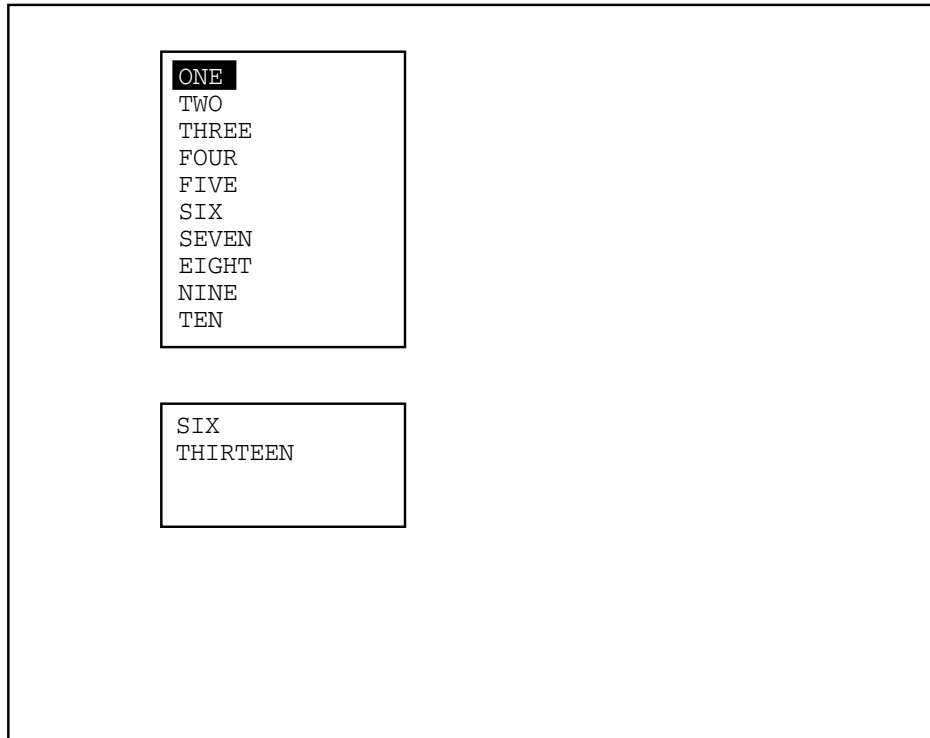


ZK-6428/1-GE

Because the menu was created with the SMG\$M\_REMOVE\_ITEM attribute, the user cannot reselect a particular menu item. However, unlike Example 1, the user can make multiple selections. In Figure SMG-52, the user has selected “SIX” and “THIRTEEN”, and the program has again highlighted the default menu item and is waiting for the user to make another selection.

# SMG\$SELECT\_FROM\_MENU

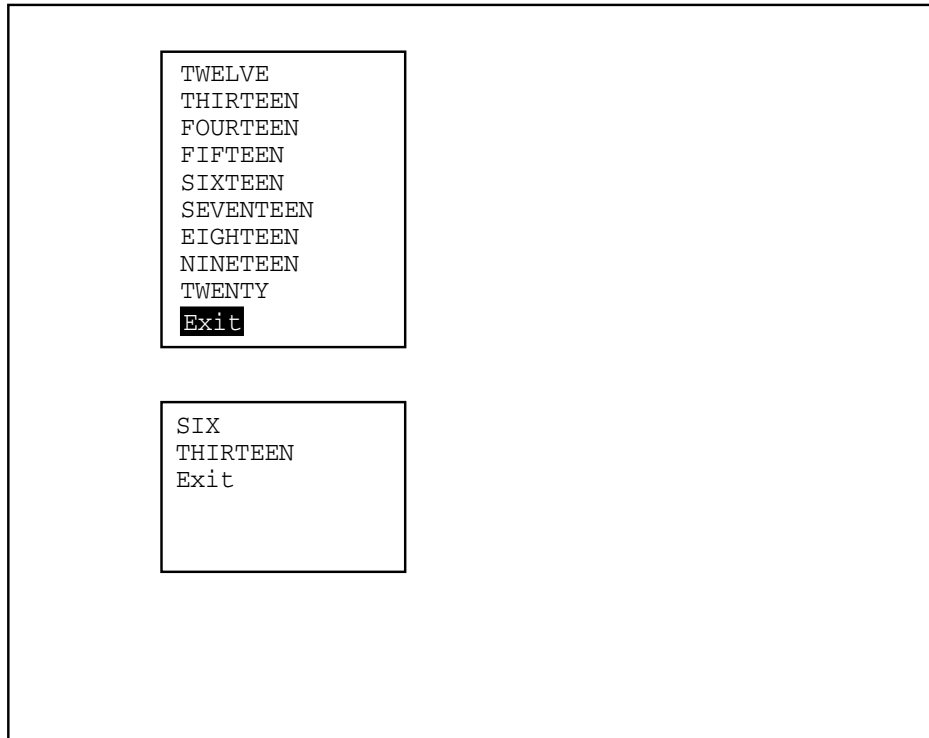
Figure SMG-52 Output Generated After Two Selections



ZK-6428/2-GE

In Figure SMG-53, the user has selected "EXIT" and the menu has been deleted, although it still appears on the screen. At this point, no more selections can be made.

Figure SMG-53 Output Generated After EXIT Is Selected



ZK-6428/3-GE

## SMG\$SELECT\_FROM\_MENU

```
3. C+
C This DEC Fortran example program illustrates the use of
C SMG$CREATE_MENU, SMG$SELECT_FROM_MENU, and SMG$DELETE_MENU
C to create an application that lets a user make multiple
C selections from a horizontal menu.
C-

 IMPLICIT INTEGER (A-Z)
 INCLUDE '($SMGDEF)'

 CHARACTER*20 c
 CHARACTER*20 a(20) /'One','Two','Three','This is Four','Five',
1 'Six','Seven','Eight','Nine','I like ten',
2 'Eleven','Twelve','Thirteen','Fourteen',
3 'Fifteen','Sixteen','Seventeen','Eighteen',
4 'Nineteen','Exit this menu.' /

 s = SMG$CREATE_PASTEBOARD(p_id)
 if (.not. s) call LIB$SIGNAL(%VAL(s))
 s = SMG$CREATE_VIRTUAL_KEYBOARD(k_id)
 if (.not. s) call LIB$SIGNAL(%VAL(s))
 s = SMG$CREATE_VIRTUAL_DISPLAY(6,50, d_id2, SMG$M_BORDER)
 if (.not. s) call LIB$SIGNAL(%VAL(s))
 s = SMG$CREATE_VIRTUAL_DISPLAY(6,50, d_id, SMG$M_BORDER)
 if (.not. s) call LIB$SIGNAL(%VAL(s))

 s = SMG$PASTE_VIRTUAL_DISPLAY(d_id2, p_id, 2,2)
 if (.not. s) call LIB$SIGNAL(%VAL(s))
 s = SMG$PASTE_VIRTUAL_DISPLAY(d_id, p_id, 10,2)
 if (.not. s) call LIB$SIGNAL(%VAL(s))

 s = SMG$CREATE_MENU(d_id,a,SMG$K_HORIZONTAL,,2,SMG$M_REVERSE)
 if (.not. s) call LIB$SIGNAL(%VAL(s))

20 s = SMG$SELECT_FROM_MENU(k_id, d_id, n,6,,,,,C,SMG$M_BOLD,0)
 if (.not. s) call LIB$SIGNAL(%VAL(s))
 s = SMG$PUT_LINE(d_id2,c)
 if (.not. s) call LIB$SIGNAL(%VAL(s))

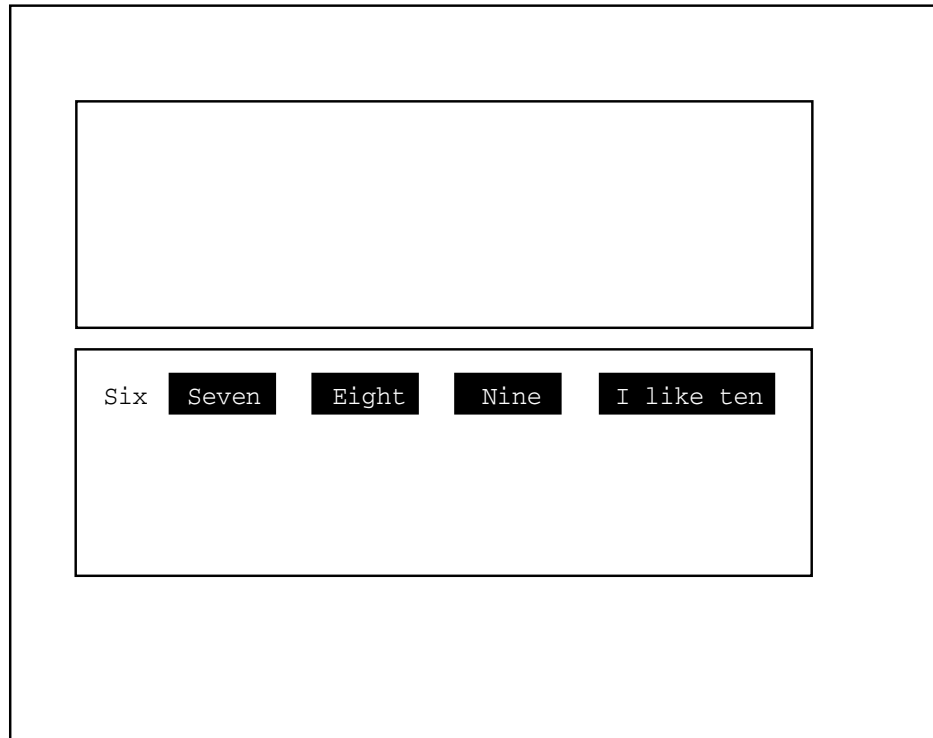
 if (n .ne. 20) goto 20

 s = SMG$DELETE_MENU(d_id)
 if (.not. s) call LIB$SIGNAL(%VAL(s))

END
```

The horizontal menu generated by this DEC Fortran example program is shown in the following figures. In Figure SMG-54, the program displays all menu items in reverse video except for the default choice. At this point, the program is waiting for the user to make a selection.

Figure SMG-54 Output Generated Before a Menu Item Is Selected

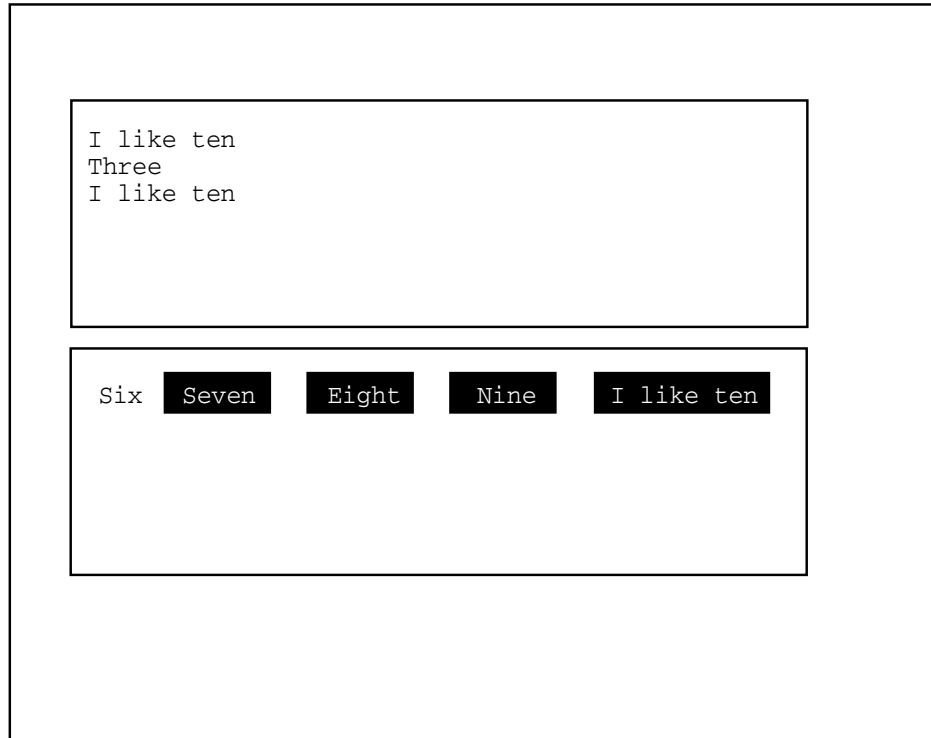


ZK-6429/1-GE

Because no attributes were specified when this menu was created, the items in the menu can be “reselected”. Figure SMG-55 shows the screen image after the user has made three selections, two of which are the same.

## SMG\$SELECT\_FROM\_MENU

Figure SMG-55 Output Generated After Three Menu Selections

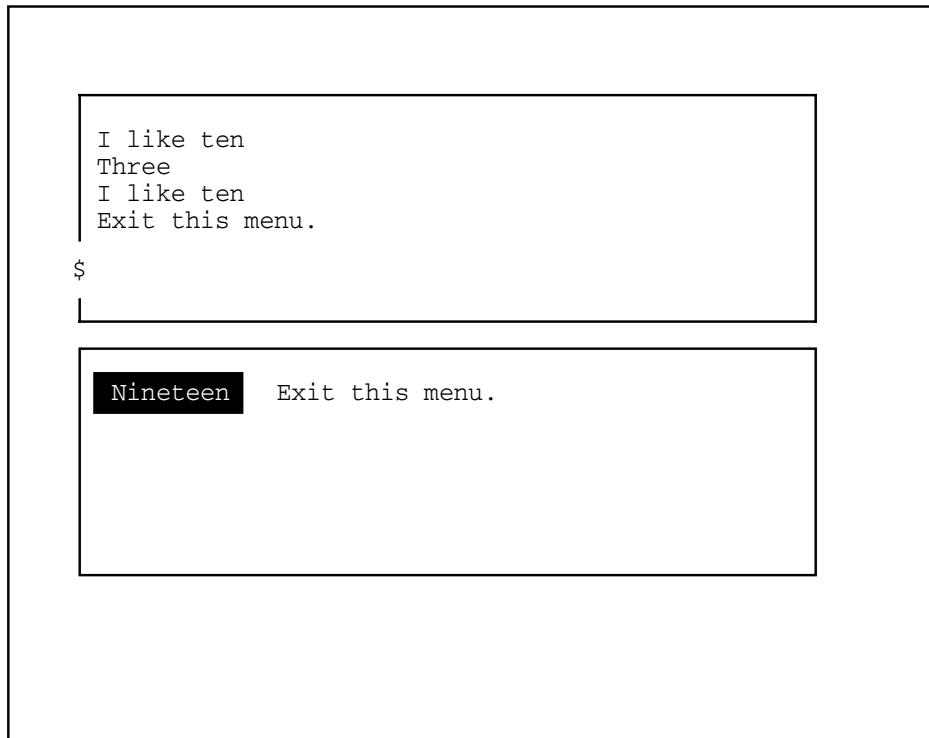


ZK-6429/2-GE

In Figure SMG-56, the user has selected “Exit this menu” and the program has completed execution.



Figure SMG-56 Output Generated After Program Completion



ZK-6429/3-GE

## SMG\$SET\_BROADCAST\_TRAPPING

---

### SMG\$SET\_BROADCAST\_TRAPPING—Enable Broadcast Trapping

The Enable Broadcast Trapping routine enables the trapping of broadcast messages.

#### Format

SMG\$SET\_BROADCAST\_TRAPPING pasteboard-id [,AST-routine] [,AST-argument]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **pasteboard-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard for which broadcast messages are to be trapped. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

##### **AST-routine**

|               |                 |
|---------------|-----------------|
| OpenVMS usage | ast_procedure   |
| type          | procedure value |
| access        | read only       |
| mechanism     | by value        |

The address of an AST routine to be called when a message is received at the pasteboard. The **AST-routine** argument is the address of the routine's procedure value.

When the **AST-routine** argument is either omitted or is given a value of 0, the BROADCAST mode is set to synchronize. In this mode, you must periodically call SMG\$GET\_BROADCAST\_MESSAGE to see if any broadcast messages have arrived.

The AST routine is called with five parameters: **AST-argument**, R0, R1, PC, and PSL (on VAX systems) or PS (on AXP systems).

## SMG\$SET\_BROADCAST\_TRAPPING

| AST Argument |
|--------------|
| R0           |
| R1           |
| PC           |
| PSL/PS       |

ZK-4803-GE

### AST-argument

OpenVMS usage user\_arg  
type longword (unsigned)  
access read only  
mechanism by value

A value to be passed to the AST routine. The **AST-argument** is an unsigned longword that contains the value to be passed to the AST routine.

### Description

SMG\$SET\_BROADCAST\_TRAPPING enables the trapping of broadcast messages sent to the specified pasteboard (terminal). When you disable broadcast trapping, any broadcast messages that have been queued to the terminal are lost. If you enable broadcast trapping with SMG\$SET\_BROADCAST\_TRAPPING but do not disable it with SMG\$DISABLE\_BROADCAST\_TRAPPING before the image exits, any messages that have been broadcast to the terminal are lost when the image exits.

The sysgen parameters DEFMBXBUFQUO and DEFMBXMXMSG are used when creating the mailbox that receives broadcast messages.

### Condition Values Returned

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                            |
| SMG\$_NOT_A_TRM | Informational message; the pasteboard is not a terminal. |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> .                           |
| SMG\$_WRONUMARG | Wrong number of arguments.                               |

Any condition values returned by \$DASSGN, \$CANCEL, or LIB\$ASN\_WTH\_MBX.

### Example

For an example using SMG\$SET\_BROADCAST\_TRAPPING, see the example for the routine SMG\$DISABLE\_BROADCAST\_TRAPPING.

---

## SMG\$SET\_CURSOR\_ABS—Set Absolute Cursor Position

The Set Absolute Cursor Position routine moves the virtual cursor to the specified position in a virtual display.

### Format

SMG\$SET\_CURSOR\_ABS display-id [,start-row] [,start-column]

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display in which to set the virtual cursor position. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row position to which the virtual cursor moves. The **start-row** argument is the address of a signed longword that contains the row number. If omitted, the cursor remains at the current row.

**start-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column position to which the virtual cursor moves. The **start-column** argument is the address of a signed longword that contains the column number. If omitted, the virtual cursor remains at the current column.

**Description**

SMG\$SET\_CURSOR\_ABS moves the virtual cursor to the specified position in the specified virtual display.

**Condition Values Returned**

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .   |
| SMG\$_INVCOL    | Invalid column.               |
| SMG\$_INVROW    | Invalid row.                  |
| SMG\$_WRONUMARG | Wrong number of arguments.    |

## SMG\$SET\_CURSOR\_MODE

---

### SMG\$SET\_CURSOR\_MODE—Set the Cursor Mode

The Set the Cursor Mode routine turns the physical cursor on or off and selects jump or smooth scrolling.

#### Format

SMG\$SET\_CURSOR\_MODE pasteboard-id ,flags

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### pasteboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Pasteboard identifier. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

##### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask that specifies scrolling and cursor attributes. The **flags** argument is the address of an unsigned longword that contains the flag. **Flags** accepts the following values:

|                      |                           |
|----------------------|---------------------------|
| SMG\$M_CURSOR_OFF    | Clears physical cursor.   |
| SMG\$M_CURSOR_ON     | Displays physical cursor. |
| SMG\$M_SCROLL_JUMP   | Jump scrolls.             |
| SMG\$M_SCROLL_SMOOTH | Smooth scrolls.           |

#### Description

SMG\$SET\_CURSOR\_MODE turns the cursor on or off and selects jump or smooth scrolling. If your terminal does not have these capabilities defined, this routine has no effect.

## Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_WRONUMARG | Wrong number of arguments.     |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SMG\$_INVARG    | Invalid argument.              |

---

## SMG\$SET\_CURSOR\_REL—Move Cursor Relative to Current Position

The Move Cursor Relative to Current Position routine moves the virtual cursor the specified number of rows and columns from the current virtual cursor position in a virtual display.

### Format

```
SMG$SET_CURSOR_REL display-id [,delta-row] [,delta-column]
```

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display in which to move the virtual cursor. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### delta-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the number of rows to move the virtual cursor. The **delta-row** argument is the address of a signed longword that contains the number of rows to move. If omitted, the virtual cursor remains at the current row position. If **delta-row** is positive, the virtual cursor moves downward the specified number of rows. If **delta-row** is negative, the virtual cursor moves upward the specified number of rows.

#### delta-column

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the number of columns to move the cursor. The **delta-column** argument is the address of a signed longword that contains the number of columns to move. If omitted, the virtual cursor remains at the current column position. If **delta-column** is positive, the virtual cursor moves the specified number of columns to the right. If **delta-column** is negative, the virtual cursor moves the specified number of columns to the left.



## Description

SMG\$SET\_CURSOR\_REL moves the virtual cursor the specified number of rows and columns relative to the current virtual cursor position. If the specified **delta-row** or **delta-column** causes the cursor to move outside the bounds of the virtual display, SMG\$\_INVROW or SMG\$\_INVCOL is returned.

## Condition Values Returned

|                 |                                                                                                              |
|-----------------|--------------------------------------------------------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                                                                                |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                                                                  |
| SMG\$_INVARG    | Invalid argument.                                                                                            |
| SMG\$_INVCOL    | An invalid value of <b>delta-column</b> caused the cursor to move outside the bounds of the virtual display. |
| SMG\$_INVROW    | An invalid value of <b>delta-row</b> caused the cursor to move outside the bounds of the virtual display.    |
| SMG\$_WRONUMARG | Wrong number of arguments.                                                                                   |

## SMG\$SET\_DEFAULT\_STATE

---

### SMG\$SET\_DEFAULT\_STATE—Set Default State

The Set Default State routine sets and/or returns the current default state for a key table.

#### Format

```
SMG$SET_DEFAULT_STATE key-table-id [,new-state] [,old-state]
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **key-table-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the key table in which you are setting or inquiring about a default state. The **key-table-id** argument is the address of an unsigned longword that contains the key table identifier.

**Key-table-id** is returned by the SMG\$CREATE\_KEY\_TABLE routine.

##### **new-state**

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | read only        |
| mechanism     | by descriptor    |

Specifies the new default state for the entire key table. The **new-state** argument is the address of a descriptor pointing to the new state string. The specified state name is converted to uppercase and stripped of trailing blanks before use.

##### **old-state**

|               |                  |
|---------------|------------------|
| OpenVMS usage | char_string      |
| type          | character string |
| access        | write only       |
| mechanism     | by descriptor    |

Receives the existing default state name of the specified key definition table. The **old-state** argument is the address of a descriptor pointing to the string into which the old state string is written.

#### Description

SMG\$SET\_DEFAULT\_STATE sets and/or returns the default state name for an entire key definition table. By changing the default state for an entire key definition table, you can use the keypad keys for a new set of functions. You can use the key definition table with the SMG\$READ\_COMPOSED\_LINE routine.

## Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_INVKTB_ID | Invalid <b>key-table-id</b> . |
| SMG\$_INVSTANAM | Invalid state name.           |
| LIB\$_INVSTRDES | Invalid string descriptor.    |

## SMG\$SET\_DISPLAY\_SCROLL\_REGION

---

### SMG\$SET\_DISPLAY\_SCROLL\_REGION—Create Display Scrolling Region

The Create Display Scrolling Region routine creates a virtual scrolling region in a virtual display.

#### Format

SMG\$SET\_DISPLAY\_SCROLL\_REGION display-id [,start-row] [,end-row]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display in which scrolling takes place. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

##### start-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the first line of the virtual scrolling region. The **start-row** argument is the address of a signed longword that contains the starting line number. If omitted, the first line of the display is used.

##### end-row

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the last line of the virtual scrolling region. The **end-row** argument is the address of a signed longword that contains the ending line number. If omitted, the last line of the virtual display is used.

## Description

SMG\$SET\_DISPLAY\_SCROLL\_REGION creates a virtual scrolling region in a specified virtual display, using the specified starting and ending lines. If the **start-row** and **end-row** arguments are omitted, the entire display becomes a scrolling region. This routine does not change the appearance of the pasteboard or the virtual cursor position.

You can define part of all of a virtual display as scrolling using SMG\$SET\_DISPLAY\_SCROLL\_REGION. Note the behavior of scrolling in the Screen Management Facility, as follows: when you call SMG\$PUT\_LINE to write to the bottom line of the scroll region, the display does not scroll up immediately. If the display scrolled up immediately, it never would be possible to use the bottom line of the scroll region. Instead, the need for a scroll is “stored” until you issue the next SMG\$PUT\_LINE call. (Note that the need for a scroll is only stored until your next call to a whole-line write routine, such as SMG\$PUT\_LINE, SMG\$PUT\_LINE\_WIDE, and so on.)

If you call another routine after SMG\$PUT\_LINE, such as SMG\$SET\_CURSOR\_ABS or SMG\$PUT\_CHARS, the stored need for a scroll will be discarded and text written to the bottom line of the scrolling region will overwrite the existing line.

## Condition Values Returned

|                 |                                                            |
|-----------------|------------------------------------------------------------|
| SS\$_NORMAL     | Normal successful completion.                              |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .                                |
| SMG\$_INVARG    | <b>End-row</b> is less than or equal to <b>start-row</b> . |
| SMG\$_INVROW    | Invalid row.                                               |
| SMG\$_WRONUMARG | Wrong number of arguments.                                 |

## SMG\$SET\_KEYPAD\_MODE

---

### SMG\$SET\_KEYPAD\_MODE—Set Keypad Mode

The Set Keypad Mode routine sets the terminal's numeric keypad to either numeric or applications mode.

#### Format

SMG\$SET\_KEYPAD\_MODE keyboard-id ,flags

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### keyboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual keyboard whose mode is to be changed. The **keyboard-id** argument is the address of an unsigned longword that contains the keyboard identifier.

**Keyboard-id** is returned by SMG\$CREATE\_VIRTUAL\_KEYBOARD.

##### flags

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Optional bit mask that specifies whether the keypad is to be in applications or numeric mode. The **flags** argument is the address of an unsigned longword that contains the flag. Valid values for **flags** are as follows:

|                               |                                     |
|-------------------------------|-------------------------------------|
| 0                             | Keypad is set to numeric mode.      |
| SMG\$M_KEYPAD_<br>APPLICATION | Keypad is set to applications mode. |

#### Description

SMG\$SET\_KEYPAD\_MODE sets the terminal's numeric keypad to either numeric or applications mode. In applications mode, numeric keypad keys are considered function keys and may be used as terminators. In numeric mode, these keys are equivalent to the corresponding keys on the main keyboard.

If the terminal does not support applications keypad mode, this routine has no effect.

**Condition Values Returned**

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_WRONUMARG | Wrong number of arguments.    |
| SMG\$_INVKBD_ID | Invalid <b>keyboard-id</b> .  |

## SMG\$SET\_OUT\_OF\_BAND\_ASTS

---

### SMG\$SET\_OUT\_OF\_BAND\_ASTS—Set Out-of-Band ASTs

The Set Out-of-Band ASTs routine either enables or disables the trapping of out-of-band control characters.

#### Format

SMG\$SET\_OUT\_OF\_BAND\_ASTS *pasteboard-id* ,*control-character-mask* ,*AST-routine* [,*AST-argument*]

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **pasteboard-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard for which out-of-band characters are enabled or disabled. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

##### **control-character-mask**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies which control characters are to be the new out-of-band control characters. The **control-character-mask** argument is the address of an unsigned longword that contains the mask. You create this mask by setting the bit that corresponds to the ASCII value of the desired character. For example, to specify that Ctrl/C (ASCII value 3) is an out-of-band control character, you set bit 3 (value 8) in the **control-character-mask**. If no bits are set in this mask, then no out-of-band ASTs occur.

##### **AST-routine**

|               |                 |
|---------------|-----------------|
| OpenVMS usage | ast_procedure   |
| type          | procedure value |
| access        | read only       |
| mechanism     | by value        |

The address of an AST routine to be called when an out-of-band control character is typed at the terminal. The **AST-routine** argument is the routine's procedure value.

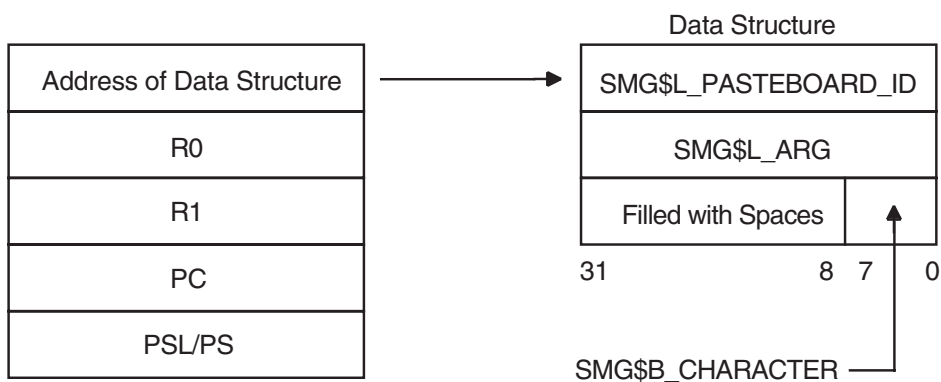


## SMG\$SET\_OUT\_OF\_BAND\_ASTS

### AST-argument

OpenVMS usage user\_arg  
type longword (unsigned)  
access read only  
mechanism by value

The argument you supply to the AST. **AST-argument** is an unsigned longword that contains the value to be passed to the AST routine. However, the AST routine may also need to determine the out-of-band character and the **pasteboard-id** at which it was typed. Therefore, the Screen Management Facility creates a three-longword structure to hold this information and passes the address of this structure as the first argument to the AST routine. The remaining four arguments are R0, R1, PC, and PSL (on VAX) and PS (on AXP). The Screen Management Facility stores the argument you supply in this structure.



ZK-4804-GE

The first longword contains the **pasteboard-id** and has the symbolic name SMG\$\_PBD\_ID. The second longword contains the **AST-argument** and has the symbolic name SMG\$\_USER\_ARG. The third longword contains the ASCII value of the out-of-band character typed and can be accessed by way of two symbolic names: SMG\$\_B\_CHAR (the low-order byte containing the ASCII value), and SMG\$\_L\_CHAR (the longword containing the ASCII value in the low-order byte and spaces in the high-order bytes).

### Description

SMG\$SET\_OUT\_OF\_BAND\_ASTS enables or disables the acceptance of out-of-band control characters at the specified terminal. If one of these characters is typed at the terminal, the AST routine is called.

This routine can be used to trap out-of-band characters, such as Ctrl/C, Ctrl/Y, and Ctrl/O.

### Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_WRONUMARG | Wrong number of arguments.     |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |

## SMG\$SET\_OUT\_OF\_BAND\_ASTS

### Example

```
!+
! This BASIC example demonstrates the use of
! SMG$SET_OUT_OF_BAND_ASTS.
!-
OPTION TYPE = EXPLICIT
OPTION CONSTANT TYPE = INTEGER

%INCLUDE "$SMGDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"
%INCLUDE "$SSDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"

EXTERNAL LONG FUNCTION SMG$CREATE_PASTEBOARD, &
 SMG$CREATE_VIRTUAL_KEYBOARD, &
 SMG$SET_OUT_OF_BAND_ASTS
EXTERNAL LONG OUT_BAND_ROUTINE

DECLARE LONG S, PASTEBOARD_ID, KEYBOARD_ID, CTRL_MASK

CTRL_MASK = (2%**SMG$K_TRM_CTRLC) + (2%**SMG$K_TRM_CTRLW) + &
 (2%**SMG$K_TRM_CTRLZ)

S = SMG$CREATE_PASTEBOARD (PASTEBOARD_ID)
IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL(S) END IF

S = SMG$CREATE_VIRTUAL_KEYBOARD (KEYBOARD_ID)
IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL(S) END IF

S = SMG$SET_OUT_OF_BAND_ASTS (PASTEBOARD_ID, &
 CTRL_MASK, &
 LOC(OUT_BAND_ROUTINE) BY VALUE, &
 KEYBOARD_ID BY VALUE)

IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL(S) END IF

SLEEP(60)

END

SUB OUT_BAND_ROUTINE(SMG$R_OUT_OF_BAND_TABLE SMG_INFO, &
 LONG R0, LONG R1, LONG PC, LONG PSL)

OPTION TYPE = EXPLICIT

%INCLUDE "$SMGDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"
%INCLUDE "$SSDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"

EXTERNAL LONG FUNCTION SMG$REPAINT_SCREEN, &
 SMG$SET_KEYPAD_MODE

DECLARE LONG S, KEYPAD_MODE

IF SMG_INFO::SMG$B_CHAR = SMG$K_TRM_CTRLC
THEN
 PRINT "CTRL/C typed"
END IF

IF SMG_INFO::SMG$B_CHAR = SMG$K_TRM_CTRLZ
THEN
 PRINT "CTRL/Z typed"
 STOP
END IF

IF SMG_INFO::SMG$B_CHAR = SMG$K_TRM_CTRLW
THEN
 S = SMG$REPAINT_SCREEN (SMG_INFO::SMG$L_PBD_ID)
 IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL(S) END IF

 KEYPAD_MODE = SMG$M_KEYPAD_APPLICATION
```

## SMG\$SET\_OUT\_OF\_BAND\_ASTS

```
S = SMG$SET_KEYPAD_MODE (SMG_INFO::SMG$L_USER_ARG, KEYPAD_MODE)
IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL(S) END IF
END IF
SUBEND
```

## SMG\$SET\_PHYSICAL\_CURSOR

---

### SMG\$SET\_PHYSICAL\_CURSOR—Set Cursor on Physical Screen

The Set Cursor on Physical Screen routine moves the physical cursor to the specified position on the pasteboard.

#### Format

SMG\$SET\_PHYSICAL\_CURSOR pasteboard-id ,pasteboard-row ,pasteboard-column

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **pasteboard-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard whose physical cursor is to move. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

##### **pasteboard-row**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the row to which the physical cursor moves. The **pasteboard-row** argument is the address of a signed longword that contains the row number.

## **pasteboard-column**

|               |                   |
|---------------|-------------------|
| OpenVMS usage | longword_signed   |
| type          | longword (signed) |
| access        | read only         |
| mechanism     | by reference      |

Specifies the column to which the physical cursor moves. The **pasteboard-column** argument is the address of a signed longword that contains the column number.

## **Description**

SMG\$SET\_PHYSICAL\_CURSOR moves the physical cursor to the specified row and column position on the specified pasteboard. This routine should not be used when pasteboard batching is in effect.

## **Condition Values Returned**

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_WRONUMARG | Wrong number of arguments.     |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SMG\$_INVARG    | Invalid column.                |

## SMG\$SET\_TERM\_CHARACTERISTICS

---

### SMG\$SET\_TERM\_CHARACTERISTICS—Change Terminal Characteristics

The Change Terminal Characteristics routine changes or retrieves the terminal characteristics for a given pasteboard.

#### Format

```
SMG$SET_TERM_CHARACTERISTICS pasteboard-id [,on-characteristics1] [,on-characteristics2]
 [,off-characteristics1] [,off-characteristics2] [,old-characteristics1]
 [,old-characteristics2]
```

#### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

#### Arguments

##### **pasteboard-id**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard whose characteristics are to be changed or retrieved. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

**Pasteboard-id** is returned by SMG\$CREATE\_PASTEBOARD.

##### **on-characteristics1**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Bit mask that specifies the terminal characteristics to be set from \$TTDEF. The **on-characteristics1** argument is the address of an unsigned longword that contains the bit mask.

##### **on-characteristics2**

|               |                     |
|---------------|---------------------|
| OpenVMS usage | mask_longword       |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Bit mask that specifies the terminal characteristics to be set from \$TT2DEF. The **on-characteristics2** argument is the address of an unsigned longword that contains the bit mask.

## SMG\$SET\_TERM\_CHARACTERISTICS

### **off-characteristics1**

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Bit mask that specifies the terminal characteristics to be reset from \$TTDEF. The **off-characteristics1** argument is the address of an unsigned longword that contains the bit mask.

### **off-characteristics2**

OpenVMS usage mask\_longword  
type longword (unsigned)  
access read only  
mechanism by reference

Bit mask that specifies the terminal characteristics to be reset from \$TT2DEF. The **off-characteristics2** argument is the address of an unsigned longword that contains the bit mask.

### **old-characteristics1**

OpenVMS usage mask\_longword  
type longword (unsigned)  
access write only  
mechanism by reference

Retrieves the current terminal characteristics in the first group. The **old-characteristics1** argument is the address of an unsigned longword that contains the bit mask.

### **old-characteristics2**

OpenVMS usage mask\_longword  
type longword (unsigned)  
access write only  
mechanism by reference

Retrieves the current terminal characteristics in the second group. The **old-characteristics2** argument is the address of an unsigned longword that contains the bit mask.

## Description

SMG\$SET\_TERM\_CHARACTERISTICS changes or retrieves the terminal characteristics for a given pasteboard. The characteristics are defined by the \$TTDEF and \$TT2DEF macro modules in Digital-supplied system symbol libraries. A benefit of using this routine is that it allows you to control multiple terminal characteristics in a single routine call.

## Condition Values Returned

|                 |                               |
|-----------------|-------------------------------|
| SS\$_NORMAL     | Normal successful completion. |
| SMG\$_NOT_A_TRM | Pasteboard is not a terminal. |
| SS\$_xyz        | Errors from LIB\$QIOW.        |

## SMG\$SET\_TERM\_CHARACTERISTICS

### Example

```
10 !+
 ! This VAX BASIC program demonstrates the use of the
 ! SMG$SET_TERM_CHARACTERISTICS routine.
 !-

 OPTION TYPE = EXPLICIT
 OPTION CONSTANT TYPE = INTEGER

 %INCLUDE "$SSDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"
 %INCLUDE "$TTDEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"
 %INCLUDE "$TT2DEF" %FROM %LIBRARY "SYS$LIBRARY:BASIC$STARLET"

 DECLARE LONG S, PASTEBOARD_ID, ON_1, ON_2, OFF_1, OFF_2, OLD_1, OLD_2

 EXTERNAL LONG FUNCTION LIB$SIGNAL(LONG BY VALUE), &
 SMG$CREATE_PASTEBOARD(LONG), &
 SMG$SET_TERM_CHARACTERISTICS(LONG, LONG, &
 LONG, LONG, LONG, LONG, LONG)

 !+
 ! Create the pasteboard
 !-

 S = SMG$CREATE_PASTEBOARD(PASTEBOARD_ID)
 IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL(S) END IF

 !+
 ! Terminal characteristics to be set
 !-

 ON_1 = TT$_M_LOWER
 ON_2 = TT2$_M_EDITING + TT2$_M_EDIT

 !+
 ! Terminal characteristics to be reset
 !-

 OFF_1 = TT$_M_WRAP + TT$_M_MECHTAB
 OFF_2 = TT2$_M_PASTHRU + TT2$_M_INSERT

 !+
 ! Change the characteristics of the terminal line associated
 ! with the pasteboard. They will be reset at image exit or when
 ! SMG$DELETE_PASTEBOARD is called. The previous characteristics
 ! are returned in OLD_1 and OLD_2.
 !-

 S = SMG$SET_TERM_CHARACTERISTICS(PASTEBOARD_ID, ON_1, ON_2, &
 OFF_1, OFF_2, OLD_1, OLD_2)
 IF S <> SS$_NORMAL THEN CALL LIB$SIGNAL(S) END IF

 IF (OLD_1 AND TT$_M_WRAP) <> 0
 THEN
 PRINT "WRAP was set"
 ELSE
 PRINT "NOWRAP was set"
 END IF

 IF (OLD_2 AND TT2$_M_ANSICRT) <> 0
 THEN
 PRINT "Pasteboard is an ANSI terminal"
 ELSE
 PRINT "Pasteboard is not an ANSI terminal"
 END IF

 END
```



---

## SMG\$UNPASTE\_VIRTUAL\_DISPLAY—Remove Virtual Display

The Remove Virtual Display routine removes a virtual display from a pasteboard.

### Format

SMG\$UNPASTE\_VIRTUAL\_DISPLAY display-id ,pasteboard-id

### Returns

|               |                     |
|---------------|---------------------|
| OpenVMS usage | cond_value          |
| type          | longword (unsigned) |
| access        | write only          |
| mechanism     | by value            |

### Arguments

#### display-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the virtual display to be removed from a pasteboard. The **display-id** argument is the address of an unsigned longword that contains the display identifier.

**Display-id** is returned by SMG\$CREATE\_VIRTUAL\_DISPLAY.

#### pasteboard-id

|               |                     |
|---------------|---------------------|
| OpenVMS usage | identifier          |
| type          | longword (unsigned) |
| access        | read only           |
| mechanism     | by reference        |

Specifies the pasteboard from which the virtual display is removed. The **pasteboard-id** argument is the address of an unsigned longword that contains the pasteboard identifier.

### Description

SMG\$UNPASTE\_VIRTUAL\_DISPLAY removes the specified display from the specified pasteboard, and thus from the screen associated with the pasteboard. This routine does not destroy the virtual display or its contents; it only removes its association with a particular pasteboard and its visibility on the screen. Any text occluded by the specified virtual display becomes visible again.

### Condition Values Returned

|                 |                                |
|-----------------|--------------------------------|
| SS\$_NORMAL     | Normal successful completion.  |
| SMG\$_INVPAS_ID | Invalid <b>pasteboard-id</b> . |
| SMG\$_INVDIS_ID | Invalid <b>display-id</b> .    |
| SMG\$_WRONUMARG | Wrong number of arguments.     |

## SMG\$UNPASTE\_VIRTUAL\_DISPLAY

SMG\$\_INVARG

Invalid argument. The specified virtual display is not pasted to the specified pasteboard.

SMG\$\_NOTPASTED

The specified virtual display is not pasted to the specified pasteboard.

## A

---

Asynchronous events, 4-1  
Argument substitutions, 5-14  
Arithmetic operations, 5-15  
AST reentrant, 4-1

## B

---

Buffering modes, 2-16

## C

---

Capability fields, 5-3  
  Boolean, 5-4  
  characters with normal ASCII value, 5-13  
  creating, 5-16  
  delimiters, 5-3  
  nonprinting characters, 5-13  
  numeric, 5-5  
  padding, 5-14  
  string, 5-6  
  used by SMG, 5-21  
  user-defined renditions, 5-12  
Character-oriented output, 2-9  
Character Set  
  ASCII, xvi, 2-16, SMG-25, SMG-60,  
    SMG-149, SMG-178, SMG-185, SMG-195,  
    SMG-225, SMG-231, SMG-235, SMG-239,  
    SMG-244, SMG-247, SMG-254, SMG-258,  
    SMG-263, SMG-266, SMG-272, SMG-275,  
    SMG-292, SMG-312, SMG-320  
  DEC Hanyu, xvi, 2-16, SMG-25, SMG-60,  
    SMG-149, SMG-178, SMG-185, SMG-195,  
    SMG-225, SMG-231, SMG-235, SMG-239,  
    SMG-244, SMG-247, SMG-254, SMG-258,  
    SMG-263, SMG-266, SMG-272, SMG-275,  
    SMG-292, SMG-312, SMG-320  
  DEC Special Graphics, xvi, 2-16, SMG-25,  
    SMG-60, SMG-149, SMG-178, SMG-185,  
    SMG-195, SMG-225, SMG-231, SMG-235,  
    SMG-239, SMG-244, SMG-247, SMG-254,  
    SMG-258, SMG-263, SMG-266, SMG-272,  
    SMG-275, SMG-292, SMG-312, SMG-320  
  DEC Supplemental, xvi  
  ISO Latin-1, xvi

## Character Set (cont'd)

  Unknown character set, SMG-149, SMG-275,  
    SMG-320  
Chinese Screen management  
  definition of, 1-1  
Composition operations, 2-1  
Cursor  
  turning on and off, SMG-360  
Cursors  
  moving, 4-3

## D

---

DEC GKS, 1-4, 4-1  
Deletion operations, 2-8  
Drawing characters, 2-11  
Drawing lines, 2-11

## E

---

Erasure operations, 2-8  
Exit handlers, 4-3

## F

---

FMS, 1-4, 4-1  
Foreign terminals  
  definition, 5-1

## G

---

GKS, 1-4, 4-1

## I

---

If state, 3-3  
Insertion operations, 2-8

## L

---

Line compositions, 3-2  
Line-oriented output, 2-9

## M

---

Menus, 2–14  
  creating, 2–14  
  deleting, 2–14  
  selecting, 2–15  
Multiple processes, 4–1

## O

---

Occlusion, 2–5  
Operational controls, 2–16  
Output operations  
  batching of, 2–17

## P

---

Pasteboards, 1–4  
Processes  
  multiple, 4–1

## R

---

Reentrant  
  AST, 4–1

## S

---

Screens  
  minimal update, 2–16  
Scroll mode  
  jump, SMG–360  
  smooth, SMG–360  
SMG\$ADD\_KEY\_DEF routine, 3–2, SMG–3  
SMG\$BEGIN\_DISPLAY\_UPDATE routine, 2–17,  
  SMG–7  
SMG\$BEGIN\_PASTEBOARD\_UPDATE routine,  
  2–17, SMG–8  
SMG\$CANCEL\_INPUT routine, 1–7, 3–1, SMG–9  
SMG\$CHANGE\_PBD\_CHARACTERISTICS  
  routine, 1–5, SMG–10  
SMG\$CHANGE\_RENDITION routine, 2–10,  
  SMG–13  
SMG\$CHANGE\_VIEWPORT routine, 2–13,  
  SMG–17  
SMG\$CHANGE\_VIRTUAL\_DISPLAY routine,  
  2–10, SMG–23  
SMG\$CHECK\_FOR\_OCCLUSION routine, 2–5,  
  SMG–26  
SMG\$CONTROL\_MODE routine, 2–16, SMG–30  
SMG\$COPY\_VIRTUAL\_DISPLAY routine,  
  SMG–33  
SMG\$CREATE\_KEY\_TABLE routine, 3–2,  
  SMG–39  
SMG\$CREATE\_MENU routine, 2–14, SMG–40

SMG\$CREATE\_PASTEBOARD routine, 1–4, 6–2,  
  SMG–46  
SMG\$CREATE\_SUBPROCESS routine, SMG–50  
SMG\$CREATE\_VIEWPORT routine, 2–13,  
  SMG–54  
SMG\$CREATE\_VIRTUAL\_DISPLAY routine,  
  1–6, SMG–58  
SMG\$CREATE\_VIRTUAL\_KEYBOARD routine,  
  1–7, 3–1, SMG–63  
SMG\$CURSOR\_COLUMN routine, 2–7, SMG–67  
SMG\$CURSOR\_ROW routine, 2–7, SMG–68  
SMG\$DEFAULT\_CHARACTER\_SET, 2–16, 3–1,  
  SMG–59, SMG–272, SMG–292  
SMG\$DEFINE\_KEY routine, 3–2, SMG–69  
SMG\$DELETE\_CHARS routine, 2–8, SMG–72  
SMG\$DELETE\_KEY\_DEF routine, 3–2, SMG–77  
SMG\$DELETE\_LINE routine, 2–8, SMG–79  
SMG\$DELETE\_MENU routine, 2–14, SMG–83  
SMG\$DELETE\_PASTEBOARD routine, 1–5,  
  SMG–84  
SMG\$DELETE\_SUBPROCESS routine, SMG–86  
SMG\$DELETE\_VIEWPORT routine, 2–13,  
  SMG–87  
SMG\$DELETE\_VIRTUAL\_DISPLAY routine,  
  1–6, 2–4, 6–2, SMG–88  
SMG\$DELETE\_VIRTUAL\_KEYBOARD routine,  
  3–1, SMG–89  
SMG\$DEL\_TERM\_TABLE routine, 5–2, SMG–71  
SMG\$DISABLE\_BROADCAST\_TRAPPING  
  routine, SMG–90  
SMG\$DISABLE\_UNSOLICITED\_INPUT routine,  
  SMG–100  
SMG\$DRAW\_CHAR routine, 2–11, SMG–102  
SMG\$DRAW\_LINE routine, 2–11, SMG–106  
SMG\$DRAW\_RECTANGLE routine, 2–11,  
  SMG–112  
SMG\$ENABLE\_UNSOLICITED\_INPUT routine,  
  4–2, SMG–117  
SMG\$END\_DISPLAY\_UPDATE routine, 2–17,  
  SMG–120  
SMG\$END\_PASTEBOARD\_UPDATE routine,  
  2–17, SMG–121  
SMG\$ERASE\_CHARS routine, 2–8, SMG–122  
SMG\$ERASE\_COLUMN routine, 2–8, SMG–127  
SMG\$ERASE\_DISPLAY routine, 2–8, SMG–129  
SMG\$ERASE\_LINE routine, 2–8, SMG–134  
SMG\$ERASE\_PASTEBOARD routine, 1–5,  
  SMG–138  
SMG\$EXECUTE\_COMMAND routine, SMG–140  
SMG\$FIND\_CURSOR\_DISPLAY routine,  
  SMG–142  
SMG\$FLUSH\_BUFFER routine, 2–16, SMG–144  
SMG\$GET\_BROADCAST\_MESSAGE routine,  
  4–2, SMG–145  
SMG\$GET\_DISPLAY\_ATTR routine, SMG–147

SMG\$GET\_KEYBOARD\_ATTRIBUTES routine, 3-2, SMG-153  
 SMG\$GET\_KEY\_DEF routine, SMG-150  
 SMG\$GET\_NUMERIC\_DATA routine, 5-2, SMG-156  
 SMG\$GET\_PASTEBOARD\_ATTRIBUTES routine, 1-5, SMG-158  
 SMG\$GET\_PASTING\_INFO routine, SMG-162  
 SMG\$GET\_TERM\_DATA routine, 5-2, SMG-164  
 SMG\$GET\_VIEWPORT\_CHAR routine, 2-14, SMG-166  
 SMG\$HOME\_CURSOR routine, 2-7, SMG-170  
 SMG\$INIT\_TERM\_TABLE routine, 5-2, SMG-172  
 SMG\$INIT\_TERM\_TABLE\_BY\_TYPE routine, 5-2, SMG-174  
 SMG\$INSERT\_CHARS routine, 2-8, SMG-176  
 SMG\$INSERT\_LINE routine, 2-8, SMG-183  
 SMG\$INVALIDATE\_DISPLAY routine, SMG-190  
 SMG\$KEYCODE\_TO\_NAME routine, 3-4, SMG-191  
 SMG\$LABEL\_BORDER routine, SMG-193  
 SMG\$LIST\_KEY\_DEFS routine, SMG-199  
 SMG\$LIST\_PASTEBOARD\_ORDER routine, 2-6, SMG-202  
 SMG\$LIST\_PASTING\_ORDER routine, 2-6, SMG-204  
 SMG\$LOAD\_KEY\_DEFS routine, 3-2, SMG-206  
 SMG\$LOAD\_VIRTUAL\_DISPLAY routine, 2-15, SMG-208  
 SMG\$MOVE\_TEXT routine, 2-3, SMG-210  
 SMG\$MOVE\_VIRTUAL\_DISPLAY routine, 2-3, SMG-214  
 SMG\$NAME\_TO\_KEYCODE routine, 3-4, SMG-216  
 SMG\$PASTE\_VIRTUAL\_DISPLAY routine, 2-1, SMG-218  
 SMG\$POP\_VIRTUAL\_DISPLAY routine, 2-4, 6-2, SMG-221  
 SMG\$PUT\_CHARS routine, 2-9, SMG-223  
 SMG\$PUT\_CHARS\_HIGHWIDE routine, 2-9, SMG-228  
 SMG\$PUT\_CHARS\_MULTI routine, 2-9, SMG-232  
 SMG\$PUT\_CHARS\_WIDE routine, 2-9, SMG-237  
 SMG\$PUT\_HELP\_TEXT routine, SMG-241  
 SMG\$PUT\_LINE routine, 2-9, SMG-245  
 SMG\$PUT\_LINE\_HIGHWIDE routine, 2-9, SMG-252  
 SMG\$PUT\_LINE\_MULTI routine, 2-9, SMG-256  
 SMG\$PUT\_LINE\_WIDE routine, 2-9, SMG-261  
 SMG\$PUT\_STATUS\_LINE routine, SMG-266  
 SMG\$READ\_COMPOSED\_LINE routine, 1-7, 3-2, SMG-268  
 SMG\$READ\_FROM\_DISPLAY routine, 2-12, SMG-274  
 SMG\$READ\_KEYSTROKE routine, 3-1, SMG-279  
 SMG\$READ\_STRING routine, 1-7, 3-1, SMG-287  
 SMG\$REMOVE\_LINE routine, 2-11, SMG-298  
 SMG\$REPAINT\_SCREEN routine, SMG-301  
 SMG\$REPASTE\_VIRTUAL\_DISPLAY routine, 2-3, SMG-305  
 SMG\$REPLACE\_INPUT\_LINE routine, SMG-311  
 SMG\$RESTORE\_PHYSICAL\_SCREEN routine, 6-3, SMG-315  
 SMG\$RETURN\_CURSOR\_POS routine, 2-7, SMG-317  
 SMG\$RETURN\_INPUT\_LINE routine, SMG-319  
 SMG\$RING\_BELL routine, SMG-323  
 SMG\$SAVE\_PHYSICAL\_SCREEN routine, 6-3, SMG-324  
 SMG\$SAVE\_VIRTUAL\_DISPLAY routine, 2-15, SMG-326  
 SMG\$SCROLL\_DISPLAY\_AREA routine, SMG-328  
 SMG\$SCROLL\_VIEWPORT routine, 2-13, SMG-331  
 SMG\$SELECT\_FROM\_MENU routine, 2-15, SMG-338  
 SMG\$SET\_BROADCAST\_TRAPPING routine, 4-2, SMG-356  
 SMG\$SET\_CURSOR\_ABS routine, 2-7, SMG-358  
 SMG\$SET\_CURSOR\_MODE routine, SMG-360  
 SMG\$SET\_CURSOR\_REL routine, 2-7, SMG-362  
 SMG\$SET\_DEFAULT\_STATE routine, SMG-364  
 SMG\$SET\_DISPLAY\_SCROLL\_REGION routine, SMG-366  
 SMG\$SET\_KEYPAD\_MODE routine, 3-2, SMG-368  
 SMG\$SET\_OUT\_OF\_BAND\_ASTS routine, 4-2, SMG-370  
 SMG\$SET\_PHYSICAL\_CURSOR routine, SMG-374  
 SMG\$SET\_TERM\_CHARACTERISTICS routine, SMG-376  
 SMG\$UNPASTE\_VIRTUAL\_DISPLAY routine, 2-2, 6-2, SMG-379  
 States, 3-3  
 Subprocesses, 4-2  
   creating, 4-2  
   deleting, 4-2  
   executing commands, 4-3

## T

---

### Terminals

- foreign
    - input support, 5-21
  - support for SET and SHOW TERMINAL commands, 5-22
  - third party
    - support for, 5-1
- Terminators, 3-3
- codes, 3-4
- TERMTABLE.EXE, 5-1, 5-16
- creating, 5-20
- TERMTABLE.TXT, 5-1, 5-16

## V

---

- VAX FMS, 1-4, 4-1
- Viewports, 1-7, 2-12
- changing characteristics, 2-14

- creating, 2-13
  - deleting, 2-13
  - moving, 2-13
  - pasting, 2-13
  - scrolling, 2-13
  - unpasting, 2-13
- Virtual displays, 1-5
- changing rendition of, 2-9
  - outputting through, 2-6
  - reading from, 2-12
  - saving, 2-15
- Virtual keyboard characteristics
- setting and retrieving, 3-2
- Virtual keyboards, 1-7
- definition of, 3-1
  - inputting through, 3-1
  - obtaining data from, 3-1

## W

---

- Writing operations, 2-8