SmartKey™

Users' Manual

Heritage Software, Inc.
2130 So. Vermont Ave.
Los Angeles, CA 90007
(213) 737-7252

COPYRIGHT NOTICE

DISCLAIMER

While considerable care has been taken to ensure that the programs forming the package conform to the specifications herein, neither **Heritage Software Inc.** nor **FBN Software** make any warranty of fitness of the products described for any purpose whatsoever.

CONTENTS

INTRODUCTION

SMARTKEY is a CP/M utility program designed to provide intelligent keyboard facilities. It is hardware independent and compatible with both Version 1.4 and Version 2.2 of the CP/M Operating System.

The program is self relocating, in a similar manner to the Digital Research program DDT, and once installed is transparent to the system user, allowing full use of all CP/M facilities and functions. By intercepting calls to the Basic I/O System (BIOS), it allows individual keys of the console keyboard to be redefined to represent different character codes from that produced by the hardware.

OVERVIEW

SMARTKEY provides the following facilities for the CP/M console user

a. The logical layout of the keyboard can be altered to suit the convenience of the user. For instance, frequently used control characters can be redefined to be represented by unused graphic or special keys, saving the time delaying use of the control shift key.

b. Characters which are not available on the particular keyboard can be allocated to unused keys. Non ASCII keyboards can be redefined to produce ASCII key codes.

c. Keys may be defined to represent a string of characters, thus providing a form of keyboard macro facility. For example, defining a key to represent the sequence '15T15L<CR>' is a simple way to implement a 'Page' command in the CP/M EDitor for consoles with a 16 line display.

All key definitions are 'dynamic' - keys can be defined, redefined, or made to revert to their hardware codes at any time, regardless of whether a program is in operation or not.

For additional ease of use, SMARTKEY may be executed with an existing set of definitions. These are still capable of being dynamically altered; the facility is provided to save repeated specification of commonly used definitions. The current set of definitions may be saved or cleared or a new set loaded without removing SMARTKEY from memory

Using the Manual

No familiarity with assembly language programming or with the details of CP/M implementation or interfacing is necessary to use SMARTKEY and the 'overview' and 'program operation' sections of this manual reflect this. In other sections, describing how the program functions, the reader is assumed to be familiar with the basics of CP/M as described in the Digital Research publications 'CP/M Interface Guide' and 'CP/M System Alteration Guide'.

Notation

In this manual, ASCII control characters are denoted by their normal ASCII abbreviation enclosed in broken brackets. For example: <CR> (carriage return), <HT> (horizontal tab). Symbolic names in lower case and enclosed in broken brackets are used to label concepts which are defined in the text. For example: <filename> (a CP/M file name), <key> (a console keyboard depression).

Distribution Disk

The following files will be found on the distributed disk:

SMARTKEY.COM	The SMARTKEY program
FIXKEY.COM	SMARTKEY utility
SKPATCH.COM	SMARTKEY installation program
FBNSUB.COM	Extended SUBMIT utility
USRPATCH.ASM	User patch area listing
CRYPTO.DEF	Sample SMARTKEY definition file
MORE.DEF	ditto
README.AAA	Stop press information

IMPORTANT: The first thing you should do after receipt of the distribution disk is to make a working copy of the files on another disk using PIP or some other utility. The original disk should then be used for backup purposes only.

Program Compatibility

SMARTKEY is compatible with any transient program which complies with the standard CP/M conventions for input-output and for determining the size of the available memory. SARTKEY is also compatible with other FBN Software 'non-transient' programs which relocate to high memory. The only constraint on using such programs simultaneously is that they must be loaded in reverse order from that in which they will be terminated. The reason for this is that a terminating program will effectively disable any program loaded 'below' itself.

Due to the non-standard system access methods employed by the Digital Research utility programs XSUB and DESPOOL, these programs are not compatible with SMARTKEY. FBNSUB, a compatible XSUB replacement is supplied with SMARTKEY. UNSPOOL, a compatible background print utility similar in functions to DESPOOL is available from FBN Software.

Further information on compatibility with specific programs and systems is given later under 'Installation'.

Memory Requirements

The residual portion of SMARTKEY, that part of the program which remains permanently in memory, occupies 2K bytes at the top of the Transient Program Area (TPA). In addition, SMARTKEY loads below the CP/M Console Command Processor (CCP) and prevents this being overlayed. The effective reduction in memory size is therefore 4K bytes. Appendix A contains a memory map showing program locations in the TPA.

PROGRAM OPERATION

Execution

SMARTKEY may be executed by typing one of the following command lines:

```
A>SMARTKEY
A>SMARTKEY <filename>
```

Where <filename> conforms to the CP/M syntax for unambiguous file names. If the three character extension is omitted it will be forced to '.DEF' by the program.

The first command line invokes the program without any keys defined. The second loads the definitions previously saved in <filename>.DEF. Examples of valid command lines are:

```
A>SMARTKEY
A>SMARTKEY MYDEFS
A>b:smartkey olddefs.def
A>SMARTKEY B:ALTDEFS
```

On execution, the program will respond with a name and version number, followed by 'READY'; the CP/M prompt 'A>' will reappear, and the system is ready for further use. Until characters are redefined, the user should be unaware of the presence of SMARTKEY, the only difference being the small reduction in available memory.

Termination

SMARTKEY will remain in operation until terminated by the operator (see FIXKEY below) or until the occurrence of a Cold Boot or Reset which reloads the entire system. Normal Warm Boots produced by the operator typing <ETX> (Control C) or by a program jumping to location 0000H will not affect the operation of the program.

Escape Character

One keyboard character is allocated as an escape character and is used to signal SMARTKEY that a key is being re-defined. This character may be altered while the program is in operation by the use of the FIXKEY utility. When SMARTKEY is in operation, the escape character cannot be used for any other purpose although, if no further definitions are required, it can be changed to a value not generated by your keyboard, thus allowing use of all the keys.

In the distributed version of the program the default value used is <ESC> (^[, ASCII code 1B). This will be the escape character when SMARTKEY is initially executed and will remain so unless it is altered by FIXKEY. SKPATCH, the interactive field-installation program for SMARTKEY allows the default value of the character to be altered and instructions are given later on how to achieve this. In the remainder of this manual, <ESC> will be used to represent the escape character.

You should ensure that the escape character you use is not required as an input character by any program you plan to use with SMARTKEY. In particular note that <ESC> should not be used with WORDSTAR as the latter uses this character as an error reset.

Key Definition

Key definitions take the following form:

<ESC><key><definition-string><ESC>

where <key> represents the key to be defined and <definition-string> is any arbitrary string of characters not including <ESC>. In other words: to define a key, type <ESC>, then the key to be defined, then the key or keys which form the definition and finally, another <ESC>

The definition has the effect of redefining <key> so that every time it is entered, the characters represented by <definition-string> are sent in succession to the program requesting input. Key definitions may be entered at any time that console input is requested, either by CP/M or by a transient program and are effective as soon as the last <ESC> has been entered.

When the <ESC> escape character is entered, SMARTKEY sends a <CR><LF> to the console. The character to be redefined is displayed on the new line, followed automatically by a space and then the characters of <definition-string> as they are typed. Control characters are displayed using the CP/M convention '^C', '^H' etc. Characters with bit 7 set (which will only appear as the defined character since SMARTKEY resets bit 7 of all characters sent to CP/M) are denoted by a leading period '.^X', '.Z' etc.

Both the escape characters and the characters of the definition are processed within SMARTKEY. The program requesting input does not see them. Instead, SMARTKEY returns a single <CR> for each key definition entered.

It is important to note that the keys forming the definition string represent their hardware codes. Definitions are not recursive! For example, if '@' is defined to represent the string 'Englebert', subsequent use of the '@' key in a definition string does NOT insert 'Englebert' into the definition.

To return a key to its original (hardware) definition, the sequence <ESC><key><ESC> may be used. (Note that <ESC><key><key><ESC> would have the same effect, but uses an additional byte of the available space for definitions). Keys may be redefined any number of times, the previous definitions are lost and the last one entered takes effect. Similarly, if you make a mistake in entering a definition, just finish it with an <ESC> and then start again.

The following are examples of key definitions:

```
DISPLAY                     INPUT STRING

A>                          <ESC>@15T15L<CR><ESC>
@ 15T15L^M
A>

A>                          <ESC>/<HT><ESC>
/ ^I
A>

A>                          <ESC>*<ESC>
*
A>
```

The first of these defines '@' as the ED 'Page' macro referred to above. The second defines '/' as a horizontal tab and the third removes any definition previously attached to '*'

Hexadecimal Input

To cater for keyboards which cannot generate all ASCII codes, a hexadecimal input mode is provided. This mode is entered by typing <ESC><ESC> which causes SMARTKEY to toggle between ASCII and HEX modes. Each time, a message giving the current mode is sent to the console. Note that this syntax precludes attempts to redefine the escape character.

In hexadecimal mode, the characters in <definition-string> comprise hexadecimal numbers in the ASCII range (00 to 7F), terminated by a non-hex character. Only the last two hex characters before the non-hex delimiter are recognized and numbers outside the 00 to 7F range are ignored so,if you make a mistake, just keep typing. SMARTKEY does not echo the characters as they are typed but, instead, echoes the character corresponding to each value entered. The console display is therefore identical to that in ASCII mode.

Appendix B contains a copy of the ASCII code giving the hexadecimal value of each character.

The following is an example of Hex mode input, defining '\' to represent <DC2>, note that the non-hex delimiter ('h' in this case) is required before the final <ESC>.

DISPLAY	INPUT STRING
A>	<ESC><ESC>
SMARTKEY: HEX Mode	
A>	<ESC>\12h<ESC>
\ ^R	
A>	

Using Saved Definitions

To save re-entering commonly used definitions each time SMARTKEY is used, the 'SMARTKEY <filename>' form of the command line may be used. This allows previously saved definitions to be loaded. Program operation is identical with that described above, with the exception that some keys will have already been defined. Definition files may also be loaded while the program is in operation as described below.

FIXKEY Utility Program

In order to minimise memory requirements for SMARTKEY, a number of less frequently used functions are contained in a separate utility program called FIXKEY. These functions are therefore available at any time that the CP/M Console Command Processor (CCP) is in operation.

FIXKEY is executed with either of the following command lines:

```
A>FIXKEY
A>FIXKEY <parameters>
```

The program initially checks that SMARTKEY is present in memory. If the first form of the command line was used, FIXKEY responds with the following menu:

```
          COMMAND MENU
0    Exit to CP/M
1    Pack and Save current definitions
2    Load a definition file
3    List contents of a definition file
4    List current definitions
5    Clear current definitions
6    Alter escape character
7    Terminate SMARTKEY
```

If SMARTKEY is not present, a message is given and all selections but 0 and 3 are disabled.

The required function may then be selected by typing the appropriate number, followed by <CR>. Selection of any of the first three functions will cause the program to prompt for the name of a definition file to be saved, loaded or listed. The user must enter a CP/M file name, eg., B:newdefs.def. If a drive specification is not included the current default drive is used. The effect of each command is listed below.

Exit to CP/M: Self explanatory. This terminates FIXKEY and returns the user to CP/M.

Pack and Save current definitions: The existing set of definitions in use by SMARTKEY is checked for consistency and, if valid, is saved under the specified file name. Before the file is saved, the space occupied by definitions which are no longer valid is recovered, thus compacting the table in memory.

Load a definition file: The contents of the specified file are checked for consistency and, if valid, are loaded into memory and replace the definitions currently in use.

List a definition file: The keys redefined in the specified file are listed together with their corresponding codes.

List current definitions: The currently redefined keys and their corresponding codes are listed.

Clear current definitions: The definition table is reinitialised, causing all currently defined keys to revert to their hardware codes.

Alter escape character: The program prompts for a new character to replace the escape character in current use.

Terminate SMARTKEY: The program is removed from core and system linkages are restored to their state prior to initial execution

The second form of the command line may be used to speed up operations and to use FIXKEY within a SUBMIT file. Use of this form suppresses display of the menu. The <parameters> are menu selections and file names, separated by spaces. The following are examples of the use of the FIXKEY <parameters> form:

Save, then clear the current set of definitions and exit to CP/M:

```
A>FIXKEY 1 frodo.def 5 0
```

Load a new definition file and exit:

```
A>fixkey 2 b:newdefs 0
```

Terminate SMARTKEY and return to CP/M:

```
A>fixkey 7 0
```

FBNSUB Utility Program

XSUB, the Digital Research 'extended submit' utility distributed with CP/M Version 2.2 is incompatible with SMARTKEY and other non-transient programs. FBNSUB, which is distributed with SMARTKEY performs an identical function to XSUB and will operate with SMARTKEY loaded.

FBNSUB is executed by including the command 'FBNSUB' in a SUBMIT file before any program reading input from the SUBMIT sequence is invoked. It will remain in memory, sending an '(FBNSUB active)' message to the console at each warm boot, until the end of the submit sequence when it is automatically terminated.

This behaviour, incidentally, is identical to that of XSUB. The CP/M User's Guide is incorrect in stating that XSUB needs to be invoked only once and will remain active past the end of a SUBMIT processing sequence.

Note that although SMARTKEY can be loaded by a SUBMIT sequence, the SMARTKEY command should appear BEFORE the FBNSUB command in the SUBMIT file. The reason for this restriction is that if FBNSUB is loaded first, it will remove SMARTKEY from memory when it is automatically terminated at the end of the SUBMIT sequence.

ADAPTING SMARTKEY

The distributed version of the program will work with any terminal and CP/M system. SMARTKEY may be adapted, however, to use additional features provided by some terminals.

Key Codes

The standard ASCII code uses only the least significant seven bits of each byte. Many keyboards, however, generate an 8 bit key code with cursor control, editing, numeric keypad or other special function keys having the high bit set. SMARTKEY can use this feature to distinguish between these and the basic keys, providing an effective increase in the number of keyboard functions available.

CP/M specifies that the high bit of each character be reset by the BIOS, limiting characters sent to CP/M to the 7 bit ASCII set. Since SMARTKEY intercepts characters sent from the BIOS, this feature will not be immediately useable unless your BIOS does not reset the high bit (in contravention of the CP/M specifications). You will therefore need to provide some method for SMARTKEY to be sent all eight bits. A number of methods for achieving this are discussed under Installation below.

Function Strings

Some terminals such as the Televideo 900 series, the Heath/Zenith 19 and others have special function keys which generate a short character string rather than a single code. SMARTKEY has the ability to detect these strings and convert them into a unique character which may then be defined in the normal way. For this feature to be used, the following conditions must be met:

a. The strings must start with the same character.
b. They must be of the same length.
c. They must have at least one character position for which each function key generates a different code.

SMARTKEY uses the initial character to recognize that a function key has been pressed and then strips off the non-significant header and trailer portions of the string. The function character is then modified (bit 7 is set). The modified character may then be used directly (SMARTKEY resets bit 7 before passing it on to the system) or redefined in the normal way.

To differentiate between the leading character of a function string and the same character typed by the operator, a timing check is used. If a second character follows the lead-in character sufficiently quickly (too quick for manual typing), SMARTKEY recognizes the sequence as the start of a string. Otherwise, the program assumes that the lead-in character has been typed manually and passes it to the system.

This system requires that a program be waiting for input before you press the function key since, if input is requested midway through a string, the lead-in character will be missed. This applies regardless of whether SMARTKEY is in use or not - it is a characteristic of the terminal. The moral is not to be too impatient when using these keys.

Examples

The following examples illustrate the definition of function keys and keys having bit 7 set. The operations are identical and differ from normal key definitions only in that a leading period indicates that the high bit of the character to be defined is set. Assume that <key1> is the F1 key of a Televideo 920C terminal which generates ^A@^M and <key2> is the numeric keypad '1' of a Xerox 820 which generates 0B1 hex.

DISPLAY	INPUT STRING
A> .@ this is F1 A>	<ESC><key1>this is F1<ESC>
A> .1 this is pad 1 A>	<ESC><key2>this is pad 1<ESC>

INSTALLATION

To perform field alterations on SMARTKEY, ensure that both SKPATCH.COM and a copy of SMARTKEY are on the default disk and then type SKPATCH. SMARTKEY should not be in memory during the installation process. SKPATCH will then prompt for the name of the file containing the SMARTKEY program and for the modifications to be made. The areas covered are listed below.

Default Escape Character

The default escape character may be altered. This is the character which initiates definitions and the default is the value when SMARTKEY is initially executed. Note that FIXKEY will still allow the value to be altered while SMARTKEY is in memory. SKPATCH prompts for the new code which may be entered directly (by hitting the required key) or as a decimal or hexadecimal value.

Function Strings

The parameters for function strings described above may be set to suit different terminals. SKPATCH asks for the initial character of the strings, the number of characters before the function character (the one which is different for each string) and the number of trailing characters. For the Televideo series, for example, the strings are of the form <SOH>c<CR> where c is the function character. The initial character is therefore ^A (<SOH>) and the length of the leader and trailer are each 1.

Eight bit Key Codes

Special input arrangements may be set up for terminals which generate 8 bit key codes. SKPATCH provides limited interactive assistance in determining the requirements for individual systems.

There are basically three options for ensuring that SMARTKEY receives eight bit codes from keyboards which have this facility. The first, which requires no modifications to SMARTKEY is to alter your CBIOS so that you may toggle between eight bit and seven bit codes, eg., by changing the value of IOBYTE. The second is to patch SMARTKEY with a jump to a monitor routine which returns the full code and the third is to patch an input routine directly into SMARTKEY. These input routines will only be called after the Console Status routine has indicated that a character is ready and thus only need to read the input port into the accumulator.

SKPATCH first checks whether 8 bit codes can be received. If not, the user is prompted for the address of a suitable monitor routine to be used for input. If no monitor routine is available, the program prompts for the address of the console input port and patches SMARTKEY to read this port when console input is required. If neither monitor nor port address are available, the adaptation is beyond the capability of SKPATCH and the program must be manually patched.

Console Status Checking

SMARTKEY normally returns 'ready' (0FF hex) to calls to the CP/M console status routine which occur while a multi-character definition is being expanded. This is compatible with most programs but a small number (notably dBase II by Ashton Tate Software) require a 'not ready' value to be returned. SKPATCH allows selection of either value so that you may create a special version for use with these programs if you require.

Warm Boot Handling

During a warm boot, the normal procedure is for the whole of CP/M with the exception of the BIOS to be reloaded from disk. This is necessary since the CCP may have been overlayed by user programs and must be re-instated. In todays systems where 48K+ is standard for memory, allowing the CCP to be overlayed is necessary for only a few giant programs -- the technique is mostly a historical remnant from the days of expensive memory when 16K was a major investment.

When SMARTKEY is in memory, the CCP cannot be overlayed and there is therefore no need to reload the system. SMARTKEY contains code which will trap warm boot requests and reinitialize CP/M without reloading. SKPATCH allows you to enable this code if you wish.

Advantages of doing this are a faster response and the prevention of system crashes when attempting to warm boot with a disk which does not contain a copy of CP/M. The possible disadvantage is the loss of some housekeeping functions which your BIOS may carry out during a warm boot so experiment with care initially.

Note that some systems which do not comply fully with the CP/M specifications may require this facility to be used. Some Superbrain and Osborne I models incorrectly reload the BIOS as well as the remainder of CP/M during a warm boot, causing SMARTKEY to appear to 'go away'. Also some versions of CP/M for NorthStar incorrectly use high memory as a scratchpad during warm boots, causing the system to hang. If you experience similar problems, you should attempt to correct them by re-installing SMARTKEY to trap and bypass warm boot requests.

User Patch Area

SKPATCH makes its modifications to a patch area of SMARTKEY initially loaded at address 0180H. A commented listing of the area is included on the SMARTKEY distribution disk in file USRPATCH.ASM. This file contains patching instructions and may be used to make additional modifications.

IMPLEMENTATION NOTES

Program Information

When initially executed, SMARTKEY is loaded into the base of the TPA by CP/M and assumes control at location 0100H. SMARTKEY then checks the size of the system and relocates itself to the top of the TPA.

Before control is returned to CP/M, the BDOS entry jump at location 0005H is altered to point to the base of SMARTKEY which, in turn, contains the BDOS jump. The reason for this is that the jump address field at 0006H is used by some programs to determine the size of the available TPA and SMARTKEY is thus protected against overwriting.

Finally, the Console Input and Console Status jumps in the BIOS are altered to point to the appropriate routines in SMARTKEY.

Definition File Structure

A standard SMARTKEY definition file is 1K bytes in length and consists of the following two sections:

a. A 512 byte Key Table, and

b. A 512 byte Definition 'Heap'.

The key table contains one word for each of the 256 possible key codes returned by the BIOS. Each word contains a pointer to the start of the definition string in the heap corresponding to that key. The pointers are relative to the start of the heap.

The first byte of the definition heap contains the escape character. Key table pointers for keys which have not been defined all point to this byte. The next 509 bytes contains definition strings, with the last (or only) byte of each string marked by having bit 7 set. The last two bytes of the heap contain a pointer to the next free space available.

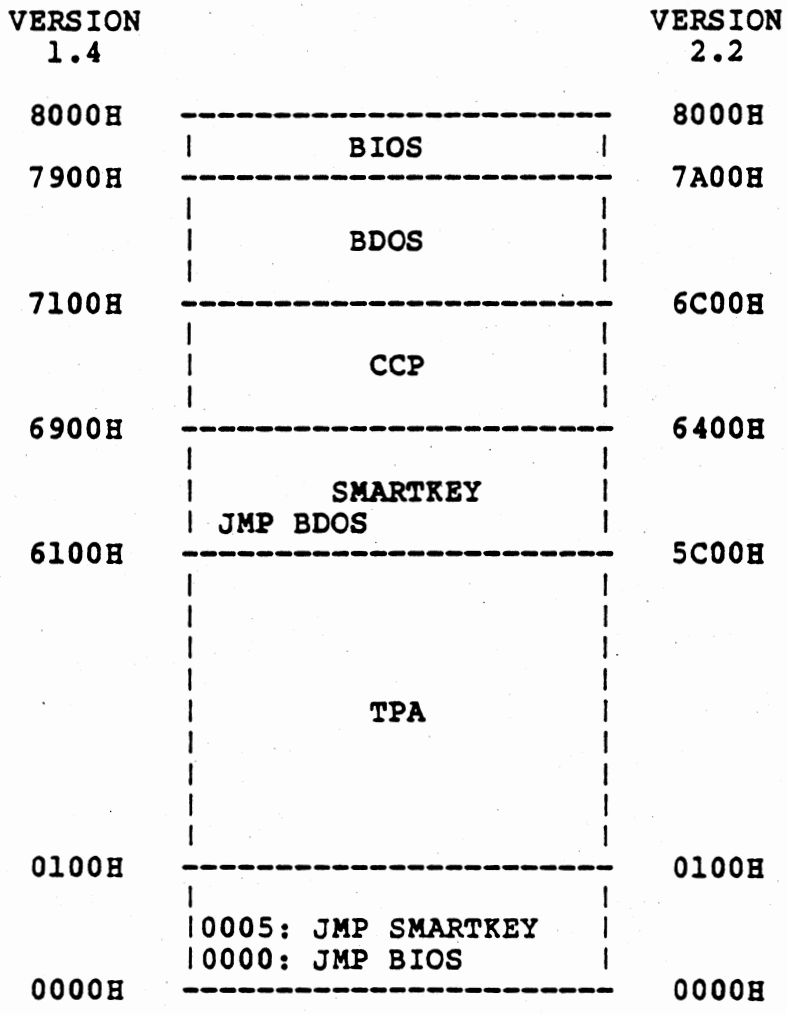
Definition strings are added to the heap in the order in which they are entered. There is no 'garbage collection' system in SMARTKEY and when the heap is full, an error message is issued. Should this happen, the FIXKEY Save function may be used to compact the table by removing definitions which are no longer valid.

APPENDIX A

System Interface

The memory map below shows the memory image in a 32K CP/M system with SMARTKEY loaded. This map shows the method used to intercept the jump to the BDOS normally located at 0005H.

32K CP/M MEMORY MAP



APPENDIX B

The ASCII Code

In the table below, the first hexadecimal digit of each character is given by the column heading and the second by the row heading. For example, 'J' is hexadecimal 4A.

	0	1	2	3	4	5	6	7
0	NUL	DLE	SPACE	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	T
5	ENG	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	VS	/	?	O	_	o	DEL

INDEX

SMARTKEY - APPLICATION NOTE 01/82

USE WITH NON-STANDARD CP/M SYSTEMS

SMARTKEY is designed to run as shipped with all standard implementations of CP/M versions 1.4 and 2.2. The program will also operate satisfactorily with some, but not all, of the so-called 'CP/M compatible' operating systems. The purpose of this note is to provide information which will allow an experienced programmer to ascertain if SMARTKEY will run with a non standard system and to provide the necessary patching details for adapting the program to do so.

A reasonable level of familiarity with the basics of CP/M and the use of DDT is assumed, as is a working knowledge of SMARTKEY and the various options available using SMKPATCH.

BIOS Jump Table

SMARTKEY alters the Warm Boot, Console Status, Console Input and Console Output jumps in the BIOS jump table and redirects control to internal routines. There are three possible problem areas here: the jump table must be used by the system itself; it must be a genuine jump table and not part of some other code and finally (although we have never run across one) a BIOS implemented in ROM will obviously not work.

In some systems, the jump table is there simply to provide CP/M compatibility and is not used by the system itself - SMARTKEY will not run satisfactorily on these systems since it is not able to intercept console input and thus has nothing to translate. TURBODOS and CDOS are two systems which are known to have a 'dummy' jump table. A simple way to check other systems is to use DDT to alter the console output jump to a RETurn instruction and check that output to the terminal ceases until the jump is corrected.

Some 64K implementations of CP/M for Northstar use two BIOS tables due to the requirement to accommodate the memory mapped disk controller at E800H. The BDOS uses one table at E700H and the second one is at the start of the BIOS at F300H. Unfortunately, the jump at location zero points to the second table and this must be altered before SMARTKEY will work successfully.

We are aware of only one instance of the second problem. In the BIOS of the NSTAR distributed multi-user system of Molecular Computer, the Console Input and Output jumps in the BIOS jump table form part of the status checking loop and thus diverting them causes a stack overflow. Before SMARTKEY can run, the loops must be altered to jump to a local address rather than back to the jump table.

The code used to intercept the BIOS jump table commences at 0357h and may be jumped out for diagnostic purposes:

```
lhld 0001h      ;get address of bios table
mvi  1,4        ;step to warm boot jump address
lxi  d,wboot    ;get address of our own routine
mov  m,e        ;   and put it in the table
inx  h
mov   m,d
dcx  h          ;redundant
lhld 0001h      ;the code above is replicated for the
...             ;constat, coninp and conout jumps
```

CCP Length

There is no way to determine the length of the CCP from within a transient program so SMARTKEY assumes a CCP length of 2K bytes (800 Hex) which is correct for CP/M 1.4 and 2.2. To save memory, if the program detects that it is running below DDT or another non-transient such as UNSPOOL, no allowance is made for the CCP. The program uses the following code, which commences at 01eah, for determining its load point:

```
    lhld 0006h      ;check whether another non-transient present
    mov  a,l        ;   bdos address is always xx06h but ddt and
    cpi  6          ;   other programs alter it to yy00h
    push psw        ;save result of test
    mvi  l,0        ;adjust load point for length of smartkey
    lxi  d,-0800h
    dad  d
    pop  psw        ;restore result of test
    jnz  .1         ;skip if not running in a 'clean' system
    lxi  d,-0800h   ;adjust load point for length of ccp
    dad  d
.1: shld loadpoint  ;and save it
```

This code must be altered in systems where the CCP/CLI length is greater than 2K. If the address of the system entry point ends in 06h, all that is necessary is to adjust the second lxi d,... instruction to allow for the new length. If not, either the cpi ... instruction can be altered to check for the correct value of the lsb of the address or the first lxi d,... can be changed to allow for both the length of SMARTKEY and the length of the CCP.

Returning to CP/M

The transient portion of SMARTKEY returns to CP/M at address 01c4h using a RST 0 instruction. To run the program under DDT for diagnostic purposes, change this to RST-7 (and dont forget to clear the default fcb by doing an i<space> instruction before g100). 01c4h may also be changed to a RET instruction to return to the CCP without doing a warm boot although this will cause problems when using SMARTKEY in a SUBMIT sequence due to a bug in CP/M.

Warm Boot Simulation

If SMARTKEY is set to trap warm boot requests and prevent reloading of the system, the program resets the default DMA address to 0080h, loads the current drive/user combination from location 0004h and returns to the CCP. For systems where the drive code at location 0004h is not implemented, the instruction: LDA 0004h at 0984h shold be NOP'ed out -- the preceding code obtains the same value via bdos calls so the c register will still be correct on entry to the CCP.

If the flag in the user patch area is set to require warm boot requests to be trapped, The CCP entry point is determined at location 0323h by the following code:

```
     lhld 0006       ;get bdos entry address
     mov  a,l        ;check whether SMARTKEY is running below
     cpi  6          ;   a non-transient program such as ddt
     jnz  .1         ;skip if it is
     mvi  l,3        ;use autoload disable entry point in ccp
     lxi  d,-800     ;adjust page # to start of ccp
     dad  d
     shld ccpentry   ;save entry point address
     ret
 .1: xra  a          ;another non-transient is present
     sta  trapflag   ;   so disable warm boot trap request
```

This may be altered as required for different ccp entry points or for systems where the bdos entry point address is other than xx06h.

System Patching

It will sometimes be necessary to make a minor patch the system to allow SMARTKEY to run -- the 64K Northstar system referred to above is an example. A convenient way to do this is to use the 16 byte area at 018ah (see USRPATCH.ASM), patching the jump at 0100h to jump to the start of this code and falling through to the beginning of SMARTKEY proper at 019ah. If this is done, location 0189h must contain a zero and SMKPATCH should not be used to try and insert a custom input routine as the patch code will be overwritten.

SMARTKEY - APPLICATION NOTE 02/82

MISCELLANEOUS PATCH INFORMATION

The following notes describe patches which may be made to SMARTKEY version 3.2 for various purposes. The patch information is given in the form of a terminal listing using DDT and was taken from an actual session using our SPOOL program to save the console output. Operator input is underlined - the comments following the ==> on the right are to describe what is going on and should not be entered.

Additional Function Keys for 'Plain Vanilla' Terminals

A number of modern terminals generate short character strings rather than single characters in response to some function and editing keys. SMARTKEY contains code to recognize these and can treat them as a single character to which a new definition can be attached. If your terminal does not generate these strings, however, this code is unused. Mike Draper of the Kingston Computer Group, Kingston, Ontario, has suggested using it to provide additional keyboard functions.

When code recognition is enabled, SMARTKEY checks each input character, looking for the 'lead in' character which marks the start of a string. Once one has been found, the program uses a timing check to distinguish between a genuine string and the lead in character typed by the operator. If the next character arrives fast enough, it's the start of a string.

By disabling the timing check, the specified character is always recognized as the start of a string. This means that you can define some character such as '\' as a lead-in character for your keyboard. When you type this, SMARTKEY will swallow it and will tag the next character. Although typing '\a' or 'a', for example, will each pass an 'a' to the system, SMARTKEY knows the difference and can redefine '\a' without affecting the use of the 'a' key. You can even retain the use of the lead-in character since typing '\\' will pass '\' to the system.

The following DDT session shows how to disable the timing check:

```
A>ddt smartkey.com              ==> load smartkey using ddt
DDT VERS 2.2
NEXT  PC
0E00 0100
-sb9b                           ==> set memory at 0B9BH
0B9B C8 0                       ==> change C8H to 00H
0B9C 3A .                       ==> enter '.' to leave set mode
-g0                             ==> back to cp/m
A>save 13 smk.com               ==> save the new program
A>
```

Now use SKPATCH to modify SMK.COM and answer 'y' to the question about terminal function keys. Specify your choice of a lead-in character (anything but <NULL>), set a header length of 1, a trailer length of 0 and the rest of the options to suit your system.

After running SMK, you should get the following response at the console - assume that '~' was specified as the lead-in character

```
input:  ~abcd~ef~~              display:  A>abcdef~
input:  <ESC>~apqr~st<ESC>      display:  A>
                                          .a pqrst
                                          A>
input:  ~abcd~ef~~              display:  A>pqrstbcdef~
```

Stopping SMARTKEY from Re-setting Bit 7

Although SMARTKEY can recognize characters with bit 7 set and distinguish them from the same character with the bit reset, it resets the bit on all characters it passes to the system. This is done to conform with the BIOS specification for Console Input (Alteration Guide P18).

Experimentation with the system, however, has indicated that it couldn't care less whether the bit was set or not and feedback from a small number of customers has shown that some commercial programs to use this bit to interact with particular terminals. The following DDT session shows how to patch SMARTKEY, to stop it re-setting bit 7 of characters it fetches from the BIOS. Note that it will still allow only characters with bit 7 reset in definition strings but will have the side effect that single character definitions will all be returned with this bit set

```
A>ddt smartkey.com
DDT VERS 2.2
NEXT  PC
0E00 0100
-0ab0
0AB0 7F ff                  ==> change ANI 7F to ANI FF
0AB1 C3 .                   ==> exit set mode with "."
-g0                         ==> back to cp/m
A>save 13 s.com             ==> save the modified program
A>
```

SMARTKEY VERSION 3.2 UPDATE
FOR VERSION 3.1 OWNERS

Version 3.2 of SMARTKEY includes the following changes:

a. INSTALL has been renamed SMKPATCH to remove conflict with the installation utilities of other packages.

b. FIXKEY now compacts the definition heap each time it saves a definition file. This removes all overwritten entries and should reduce the incidence of definition table overflows.

c SMARTKEY has provision to prevent the system from reloading CP/M during each warm boot. This should be satisfactory in most systems and is required in some NorthStar, SuperBrain and Osborne I systems which do not comply fully with the CP/M specifications. This option may be set using SMKPATCH.

d. SMKPATCH also allows the option of having SMARTKEY return 'not ready' in response to a console status request while a definition is being expanded instead of the default value of 'ready'. This change is necessary to use SMARTKEY successfully with dBase II which otherwise ignores some characters in definitions.

e. Users with Televideo type terminals no longer have to specify a timing constant when installing SMARTKEY to use the function-key strings. The program is now able to recognize these keys within a wide range of processor and terminal speeds.

Definition files made with previous versions are compatible with version 3.2 of SMARTKEY. The associated utilities (FIXKEY, INSTALL and SMKPATCH) however, are only compatible with the SMARTKEY version they were supplied with. We recommend that you archive the earlier versions and use version 3.2 exclusively.