SVC_COUNT()

Provides access to the diagnostics statistical counts (where implemented, as defined on the following page).

Prototype

```
#include <txosvc.h>
int SVC_COUNT( operation, counter );
int operation, counter;
```

Parameters

`counter` specifies which count is affected.

The `operation` parameter determines the action to be taken; operation may have the following values:

0	Set the designated counter to zero. No meaningful result is returned.
1	Increment the designated counter (by one). Only counters 0 ... 19 may be modified by the user. No meaningful result is returned.
2	Return the value of the designated counter.

Returns

`count` If `operation` is 2, `SVC_COUNT()` returns the value of the count selected by `counter`.

If `operation` = 0 or 1, no meaningful value is returned.

Example

See page 11-116.

Diagnostics Statistical Counts

The terminal maintains a collection of user-defined and operating system statistical counts for diagnostic evaluation. These counts are accumulated until a command is issued by the diagnostics software to clear them. Note that some counts, (for example LAN counts or counts on "intellegent" com ports) may not be available to all platforms

Counts 0 to 19 are reserved for user applications.

Count.....	Description
1–19.....	User-defined counts
20	Physical key presses
21	Power-on cycles or system restarts
22	System boots (exiting from system mode via [CLEAR] key)
23	ZONTALK downloads [0] attempted
24	Direct uploads [*] attempted
25	Direct downloads [#] attempted
26	Normal beeps
27	Error beeps
28	Bar code reads attempted
29	Bar code reads with errors
30	Card reads attempted
31	Card reads with errors
32	# of OK modem message
33	# of CONNECT (at 300 baud) modem messages
34	# of RING modem messages
35	# of NO CARRIER modem messages
36	# of ERROR modem messages
37	# of CONNECT 1200 modem messages
38	# of NO DIALTONE modem messages
39	# of BUSY modem messages
40	(reserved)

Count.... Description

41 ..... # of NO LINE modem messages
VeriFone addition to Hayes messages

42 ..... ACKs sent by terminal on the modem port†

43 ..... NAKs sent by terminal on the modem port†

44 ..... ACKs received by terminal on the modem port†

45 ..... NAKs received by terminal on the modem port†

46 ..... ACKS sent on COM1 port†

47 ..... NAKS sent on COM1 port†

48 ..... ACKS received on COM1 port†

49 ..... NAKS received on COM1 port†

50 ..... ACKS sent on PIN Pad/Bar Code port†

51 ..... NAKS sent on PIN Pad/Bar Code port†

52 ..... ACKS received on PIN Pad/Bar Code port†

53 ..... NAKS received on PIN Pad/Bar Code port†

54 ..... ACKS sent on COM3 port†

55 ..... NAKS sent on COM3 port†

56 ..... ACKS received on COM3 port†

57 ..... NAKS received on COM3 port†

58 ..... Number of DIALING COMPLETE modem messages
VeriFone extension to Hayes messages

59 ..... ACKS sent on LAN port†

60 ..... NAKS sent on LAN port†

61 ..... ACKS received on LAN port†

62 ..... NAKS received on LAN port†

63 ..... Transmit timeouts on LAN port

64 ..... Runt packets‡received on LAN

65–68 ... Reserved for future use

†ACK/NAK counts received or sent are available only to terminals with drivers supporting "intelligent" protocols, such as VISA protocol, Peer-to-Peer LAN protocol, an intelligent PIN Pad protocol, etc.

‡Runt packet: Header is less than minimum required size and is invalid (not processed).

Example

```
/* COUNT.C */

#include <stdio.h>
#include <txosvc.h>

int i;
main () {

   /* Reset user counter 10 to 0 */
   SVC_COUNT(0,10);

   /* Increment the counter 1000 times. */
   for (i=0; i<1000; i++) SVC_COUNT(1,10);

   /* Display the counter value. */
   printf("\fCOUNTER 10=%d",SVC_COUNT(2,10));
}
```

SVC_CRC_CALC()

Calculates a Cyclic Redundancy Check (CRC).

Prototype

```
#include <txosvc.h>
unsigned int SVC_CRC_CALC( type, buffer, size );
char type;
int size;
char *buffer;
```

Parameters

The CRC is performed on data pointed to by `buffer` of length `size`.

The `type` parameter specifies what type of calculation is to be employed:

0	*Longitudinal Redundancy Check*. An exclusive OR (XOR) is performed on each byte in the buffer. The resulting LRC is stored in the lower byte of the integer; the upper byte is set to zero.
1	*Cyclic Redundancy Check* (CRC16), based on the standard CRC16 polynomial $X^{16}+X^{15}+X^2+1$. Bits are read least-significant-bit first, as is traditional in hardware implementations. The CRC1 value is returned in the low byte and the CRC2 value is returned in the high byte.
2	*CRC16, most-significant-bit first*. The CRC1 value is returned in the high byte and the CRC2 value is returned in the low byte.
3	*CCITT polynomial*, $X^{16}+X^{12}+X^5+1$, lsb-first
4	*CCITT polynomial, msb-first*

Returns

`crc` The CRC value of the data stored in `buffer`.

Notes

To communicate with Direct Load (DL.EXE) or ZONTALK, use `type` = 2.

For both CRC16 computations, the initial value used for the checksum is reset to all zeros (0x0000). For both CCITT

computations, the initial value used for the checksum is set to all ones (0xFFFF).

Example In the example code below, SVC_CALC_SVC() performs a CRC 16-LSB on the string "VeriFone, Inc." If each of the four types of CRCs were specified, the bytes appended to the printf string would be:

CRC16-LSB:	4B 3C
CRC16-MSB:	D5 25
CCITT-LSB:	DF 2A
CCITT-MSB:	F6 03

```
/* CRC_CALC.C */

char buffer[20];
unsigned int crc;
main() {

  /* Build buffer where \03 appends <ETX> */
  strcpy(buffer, "VeriFone, Inc.\03");

  /* Get and display the CRC. */
  crc = SVC_CRC_CALC(1, buffer, strlen(buffer));
  printf("\fCRC16-LSB=%02x,%02x",crc&0x00FF,
          crc>>8);
}
```

SVC_DSP_2_HEX()

Converts ASCII-hex "display" characters to hexadecimal.

Prototype

```
#include <txosvc.h>
void SVC_DSP_2_HEX( hextext, hexbuff, count );
char *hextext, *hexbuff;
int count;
```

Parameters

`hextext` is an array of ASCII-hex "display" characters in the range `0x30 - 0x39` ("0" – "9") and `0x41 - 0x46` ("A" – "F"). The number of characters in the string must be divisible by two.

`hexbuff` is the location where the binary result is stored.

`count` is the number of pairs of ASCII-hex bytes to be converted, e.g., the length of the string divided by two.

Notes

Each pair of ASCII-hex characters is converted to a single binary byte value. The size of `hexbuff` should be large enough to accommodate the result. For example, if four ASCII-hex characters are in `hextext`, the value in `count` should be 2, and therefore two bytes are stored into `hexbuff`.

Returns

VOID

Related

`SVC_HEX_2_DSP()`

Example

See following page.

Example

```
/* DSP2HEX.C */

#include <stdio.h>
#include <strings.h>
#include <txosvc.h>

char dsp[10],hex[3];
main() {

   /* Load "12345F" into dsp */
   strcpy(dsp,"12345F");

   /* Convert to hex and display */
   SVC_DSP_2_HEX(dsp,hex,3);
   printf("\f%x%x%x",hex[0],hex[1],hex[2]);
}
```

SVC_GET_UNIT_ID()

Retrieves the terminal unit identification number.

Prototype

```
#include <txosvc.h>
void SVC_GET_UNIT_ID( cstr );
char *cstr;
```

Parameters `cstr` is the location to return the unit ID.

Notes Some countries require each terminal to have a unique *unit identification number*. This command retrieves this number from the system, storing it as a string (if used), or by writing 12000000 on terminals that do not have a unit ID.

Returns The `cstr` array is filled with a 9-byte counted string: 0x09, "mmxxxxxx",

where "mm" is the manufacturer's ID (normally "12" for VeriFone), and

"xxxxxx" is either 000000—unit ID not used—or the terminal's unit ID number.

Example

```
/* UNIT_ID.C */

#include <stdio.h>
#include <txosvc.h>

char buffer[10];
main() {
   /* Get the unit id and display. */
   SVC_GET_UNIT_ID(buffer);
   printf("\fUNIT ID %s",buffer+1);
}
```

SVC_HEX_2_DSP()

Converts hexadecimal data to an array of ASCII-hex "display" characters.

Prototype

```
#include <txostd.h>
void SVC_HEX_2_DSP( hexbuff, hextext, count );
char *hexbuff, *hextext;
int count;
```

Parameters

`hexbuff` contains an array of binary byte values.

`hextext` is the location where the converted ASCII-hex "display" characters in `0x30...0x39` ("0"..."9") and `0x41...0x46` ("A"..."F") are stored.

`count` specifies the number of bytes in `hexbuff`.

Returns VOID

Related `SVC_DSP_2_HEX()`

Example

```
/* HEX2DSP.C */

#include <stdio.h>
#include <txosvc.h>

char dsp[10],hex[3];
main() {

   /* Load "12345F" into hex */
   hex[0]=0x12; hex[1]=0x34; hex[2]=0x5F;

   /* Convert to dsp and display */
   SVC_HEX_2_DSP(hex,dsp,3);
   dsp[7]=0; /* Null termination */
   printf("\f%s",dsp);
}
```

SVC_INFO_BAR()

Returns bar code reader information.

Prototype

```
#include <txosvc.h>
int SVC_INFO_BAR (feature);
int feature;
```

Parameters The `feature` parameter determines the type information to be returned, as follows:

0	Return type of bar code reader installed
1	Returns bar code encoding type (Raw timing, Code39, UPC, etc.)

Returns Return values are based on the value of `feature`:

feature =	Return Values
0	0 = No bar code reader installed
	1 = Bar code wand installed
	2 = Bar code slot installed
1	0 = Raw timing
	1 = Code39
	2 = EAN/UPC
	3 = 2 of 5

Note This function is supported on EPROM Version 10 and higher.

Example See following page.

Example

```
/* BAR.C */

#include <stdio.h>
#include <config.h>
#include <txosvc.h>
int decoding;

main () {

   /* Display the type of barcode device supported. */
   switch (SVC_INFO_BAR(0)) {
      case 0: write(STDOUT, "\fNO BARCODE", 11); break;
      case 1: write(STDOUT, "\fBARCODE WAND", 13); break;
      case 2: write(STDOUT, "\fBARCODE SLOT", 13); break;
   }
   getchar();

   /* Get the forms of barcode support. */
   decoding = SVC_INFO_BAR(1);

   /* Is raw timing supported? */
   if (decoding & 1) {
      write(STDOUT, "\fRAW TIMING", 11);
      getchar();
   }

   /* Is Code 3 or 9 supported? */
   if (decoding & 2) {
      write(STDOUT, "\fCODE 3 OF 9", 12);
      getchar();
   }

   /* Is EAN/UPC supported? */
   if (decoding & 4) {
      write(STDOUT, "\fEAN/UPC", 8);
      getchar();
   }

   /* Is 2 of 5 supported? */
   if (decoding & 8) {
      write(STDOUT, "\f2 OF 5", 7);
      getchar();
   }
}
```

SVC_INFO_CARD()

Returns card reader information.

Prototype

```
#include <txostd.h>
int SVC_INFO_CARD (feature);
int feature;
```

Parameters `feature` must be set to zero before a call to this function is made.

Returns

`bits` With `feature` set to zero, the following bit map of supported tracks is returned (bit 0 is low order bit):

Bit	Value	Card Reader
0	0x01	Dual track
1	0x02	Track 1
2	0x04	Track 2
3	0x08	Track 3
4	0x10	Triple track
5	0x20	JIS support for track 3 (Japanese Industrial standard)

❖ *If bit 5 (JIS support) is set, it will be set in conjunction with either bit 0 (dual track), or bit 4 (triple track).*

`-1` Terminal does not have a card reader.

Note This function is supported on EPROM Version 10 and higher.

Example See following page.

Example

```
/* CARD.C */

#include <stdio.h>
#include <config.h>
#include <txosvc.h>
int decoding;

main () {

   /* Get the form of card reading support. */
   decoding = SVC_INFO_CARD(0);

   /* Is a dual track card reader supported? */
   if (decoding & 1) {
      write(STDOUT, "\fDUAL TRACK", 11);
      getchar();
   }

   /* Is Track 1 the only supported track? */
   if (decoding & 2) {
      write(STDOUT, "\fTRACK 1 ONLY", 13);
      getchar();
   }

   /* Is Track 2 the only supported track? */
   if (decoding & 4) {
      write(STDOUT, "\fTRACK 2 ONLY", 13);
      getchar();
   }

   /* Is Track 3 the only supported track? */
   if (decoding & 8) {
      write(STDOUT, "\fTRACK 3 ONLY", 13);
      getchar();
   }
}
```

SVC_INFO_DSP()

Returns information about the display device.

Prototype

```
#include <txosvc.h>
int SVC_INFO_DSP (feature);
int feature;
```

Parameters The type of information requested is passed in the `feature` parameter as follows:

Value	Display Feature
0	Characters per row
1	Characters per column
2	Pixels per row
3	Pixels per column
4	Packed tail characters (comma period)
5	Number of fonts supported
6	Display type
7	Adjustable contrast

Returns The following values are returned based on the parameter passed in `feature`:

feature =	Returned Value
0	Characters per row, 0 if variable
1	Characters per column, 0 if variable
2	Pixels per row, 0 if not pixel-based
3	Pixels per column, 0 if not pixel-based
4	Packed tail characters, 0=no, 1=yes If yes, remember that commas/periods may be placed on the same character cell as the preceding character.
5	Number of fonts supported, 0 if variable
6	0 = segment-based, 1 = pixel-based
7	Adjustable contrast (0=no, 1=yes)

Note This function is supported on EPROM Version 10 and higher.

Example See following page.

Example

```
/* DSP.C */

#include <stdio.h>
#include <txosvc.h>

main () {

   /* Show display dimensions in terms of pixels or chars. */
   if (SVC_INFO_DSP(6))
      printf("\f%d x %d PIXELS", SVC_INFO_DSP(2), SVC_INFO_DSP(3));
   else
      printf("\f%d x %d CHARS", SVC_INFO_DSP(0), SVC_INFO_DSP(1));
   getchar();

   /* Show number of fonts.  0 means a variable number. */
   if (SVC_INFO_DSP(5))
      printf("\f%d FONT(S)", SVC_INFO_DSP(5));
   else
      printf("\fVARIABLE # FONTS");
   getchar();

   /* Are tail characters (..) packed into same */
   /* char space as previous characters?        */
   printf("\fPACKED TAILS=");
   if (SVC_INFO_DSP(4))
      printf("YES");
   else
      printf("NO");
   getchar();

   /* Is the contrast of the display adjustable? */
   printf("\fADJUST CONT=");
   if (SVC_INFO_DSP(7))
      printf("YES");
   else
      printf("NO");
}
```

SVC_INFO_EPROM()

Returns EPROM (firmware) identification and version number.

Prototype

```
#include <txosvc.h>
void SVC_INFO_EPROM (buffer);
char *buffer;
```

Parameters None

Returns The EPROM identification string is returned as a counted string in buffer.

Related Same as SVC_VERSION().

Note This function is supported on EPROM Version 10 and higher.

Example

```
/* EPROM.C */

#include <stdio.h>
#include <txosvc.h>

char buffer[10];
main() {

   /* Get the EPROM id. */
   SVC_INFO_EPROM(buffer);

   /* Null terminate the string and display. */
   buffer[buffer[0]]=0;
   printf("\fEPROM=%s", &buffer[1]);
}
```

SVC_INFO_KEY()

Returns information about the keyboard.

Prototype

```
#include <txosvc.h>
int SVC_INFO_KEY (feature);
int feature;
```

Parameters The `feature` parameter specifies the type of information this function requests per one of the following values:

0 Number of standard keys
1 Number of function keys which are not screen-addressable
2 Number of functions keys which are screen-addressable.
3 Calculator-style vs. Telco-style

Returns Return values are based on the value of `feature`:

feature =	Returned Value
0	Number of standard keys (0-16)
1	Number of non-screen-addressable function keys (0-4)
2	Number of screen addressable function keys (0-4)
3	0 = Telco-style 1 = Calculator-style

Note This function is supported on EPROM Version 10 and higher.

Example

```
/* INFO_KEY.C */

#include <stdio.h>
#include <txosvc.h>

main () {

   /* Display type of keypad layout. */
   if (!SVC_INFO_KEY(3))
      printf("\fTELCO KEYPAD");
   else
      printf("\fCALC KEYPAD");
   getchar();

   /* Display number of standard keys. */
   printf("\f%d STANDARD KEYS", SVC_INFO_KEY(0));
   getchar();

   /* Display number of screen addressable keys. */
   printf("\f%d SCREEN KEYS", SVC_INFO_KEY(2));
   getchar();

   /* Display number of non-screen addressable keys. */
   printf("\f%d EXTRA KEYS", SVC_INFO_KEY(1));
}
```

SVC_INFO_PLATFORM()

Returns amount of static RAM in the terminal.

Prototype

```
#include <txosvc.h>
int SVC_INFO_PLATFORM (feature);
int feature;
```

Parameters The `feature` parameter only accepts a value of 3, which returns the amount of static RAM (in kilobytes) installed on the terminal.

Returns RAM size.

Note This function is supported on EPROM Version 10 and higher.

Related Same as `SVC_RAM_SIZE()` with `feature` parameter added.

Example

```
/* PLATFORM.C */

#include <stdio.h>
#include <txosvc.h>

main () {

   /* Display the RAM size. */
   printf("\f%dK BYTES RAM", SVC_INFO_PLATFORM(3));
}
```

SVC_INFO_PTID()

Returns the permanent terminal identifier (also known as Unit identification number).

Prototype

```
#include <txosvc.h>
void SVC_INFO_PTID (buffer);
char *buffer;
```

Parameters `buffer` must be at least 9 bytes in length to hold the terminal identifier.

Returns `buffer` is a 9-byte counted string in the following format:

0x09, mmxxxxxx

where "mm" is the manufacturer's ID (normally "12" for VeriFone), and

"xxxxxx" is either 000000—unit ID not used—or the terminal's unit ID number.

Note This function is supported on EPROM Version 10 and higher.

Related Same as `SVC_GET_UNIT_ID()`.

Example

```
/* PTID.C */

#include <stdio.h>
#include <txosvc.h>

char buffer[10];
main() {

   /* Display the permanent terminal id (unit id). */
   SVC_INFO_PTID(buffer);
   printf("\fPTID %s", buffer+1);
}
```

SVC_INFO_TYPE()

Returns information about the type of processor and terminal.

Prototype

```
#include <txostd.h>
void SVC_INFO_TYPE (feature);
int feature;
```

Parameters The `feature` parameter specifies they type of information to return, as follows:

0 Type of processor (CPU)
1 Type of terminal

Note This function is supported on EPROM Version 10 and higher.

Returns The following values are returned based on the value passed in `feature`:

feature =	Returned Value
0	0 = Z80
	1 = Z180
	2 = 68301
1	0 = OMNI 330
	1 = OMNI 380
	2 = OMNI 385 LAN
	3 = OMNI 390
	4 = OMNI 460
	5 = OMNI 480
	6 = OMNI 490
	7 = OMNI 395
	8 = OMNI 395 LAN
	9 = OMNI 385

Note This function is supported on EPROM Version 10 and higher.

Example

```
/* TYPE.C */

#include <stdio.h>
#include <config.h>
#include <txosvc.h>

main () {

   /* Display the processor type. */
   switch (SVC_INFO_TYPE(0)) {
      case 0: write(STDOUT, "\fZ80 CPU", 8); break;
      case 1: write(STDOUT, "\fZ180 CPU", 9); break;
      case 2: write(STDOUT, "\f68301 CPU", 10); break;
   }
   getchar();

   /* Display the terminal type. */
   switch (SVC_INFO_TYPE(1)) {
      case 0: write(STDOUT, "\fOMNI 330", 9); break;
      case 1: write(STDOUT, "\fOMNI 380", 9); break;
      case 2: write(STDOUT, "\fOMNI 385 LAN", 13); break;
      case 3: write(STDOUT, "\fOMNI 390", 9); break;
      case 4: write(STDOUT, "\fOMNI 460", 9); break;
      case 5: write(STDOUT, "\fOMNI 480", 9); break;
      case 6: write(STDOUT, "\fOMNI 490", 9); break;
      case 7: write(STDOUT, "\fOMNI 395", 9); break;
      case 8: write(STDOUT, "\fOMNI 395 LAN", 13); break;
      case 9: write(STDOUT, "\fOMNI 385", 9); break;
   }

}
```

SVC_INT2()

Converts a binary number to ASCII-decimal.

Prototype

```
#include <txosvc.h>
void SVC_INT2( number, destination );
unsigned int number;
char *destination;
```

Parameters `number` is converted to the equivalent ASCII-decimal string.

Returns VOID

Notes The string returned in `destination` is a counted string. `destination` should be large enough to hold the longest expected string plus one byte for the count.

Example

```
/* INT2.C */

#include <stdio.h>
#include <txosvc.h>

char destination[10];
main() {

   /* Convert a number to a string. */
   SVC_INT2(1234, destination);

   /* Null terminate the string and display. */
   destination[destination[0]]=0;
   printf("\f%s",&destination[1]);
}
```

SVC_KEY_NUM()

Uses keyboard, display, and beeper input to get a formatted decimal number (counted string).

Prototype

```
#include <txosvc.h>
int SVC_KEY_NUM( dest_buff, max_digits,
                 fraction, punctuate );
char *dest_buff;
int max_digits, fraction, punctuate;
```

Parameters

`dest_buff` specifies where the resulting counted string of digits is stored.

`max_digits` (must be less than 15) specifies the maximum number of digits permitted (excluding punctuation marks).

`fraction` (must be less than 10) specifies the number of digits to the right of the radix, e.g., the fractional value.

`punctuate` specifies the style of punctuation requested:

0 – Radix=point	No separator	12345678.90
2 – Radix=point	Separator=comma	12,345,678.9
4 – Radix=comma	No separator	12345678,90
6 – Radix=comma	Separator=point	12.345.678,90

Returns

`bytes_read` Integer value containing the actual number of bytes stored in the destination buffer (including puncutuation characters) on a successful read.

If the user presses [ENTER] without additional input, the destination buffer contains a counted string of zeros in the format specified by `fraction`.

if `fraction` = 0:
`bytes_read` = 2, `buffer` = 0.

if `fraction` = 1:
`bytes_read` = 3, `buffer` = 0.0

if `fraction` = 2:
`bytes_read` = 4, `buffer` = 0.00

-1 The user has pressed the [CLEAR] key to cancel the read operation.

Notes The [CLEAR] key trap (Trap 32, as described in *Chapter 7) is disabled when this function is called.*

The [ALPHA] key may be used to toggle between positive and negative numbers. A negative number is displayed with leading sign (-).

The [BACKSPACE] key may be used to delete the last character entered.

The [CLEAR] key may be used to abort the read.

Example

```
/* KEY_NUM.C */

#include <stdio.h>
#include <txosvc.h>

char dest_buff[20];
main() {

   printf("\fENTER 123456789");

   /* Display should look like '12,345,678.9' after digits
      are entered. */
   SVC_KEY_NUM(dest_buff, 9, 1, 2);

   /* Null terminate the string and display. */
   dest_buff[dest_buff[0]]=0;
   printf("\fSTRING=%s",&dest_buff[1]);
}
```

SVC_KEY_TXT()

Uses keyboard, display and beeper to get formatted data input.

Prototype

```
#include <txosvc.h>
int SVC_KEY_TXT( dest_buff, type,
                 max_allowed,
                 min_allowed );
char *dest_buff;
int type, max_allowed, min_allowed;
```

Parameters

`dest_buff` is the location for the entered data to be returned (should be large enough to receive `max_allowed` bytes plus one count byte.)

`type` specifies the type of data entry allowed.

- 0 Numeric keys only
- 1 Alphanumeric keys
- 2 Numeric keys for password. Each key is echoed with an asterisk.
- 3 Alphanumeric keys for password. Each key is echoed with an asterisk.

`max_allowed` is the maximum number of bytes that can be entered.

`min_allowed` is the minimum number of bytes allowed.

Returns

`bytes_read`	Integer value containing the actual number of bytes read on a successful read.
`-1`	The user has pressed the [CLEAR] key to cancel the read operation.

Notes

The [CLEAR] key trap (Trap 32, described in *Chapter 7)* is disabled when this function is called.

The control keys in the fourth column of the keyboard ([CLEAR], [BACKSPACE], [ALPHA], [ENTER]) are used to provide primitive editing options.

The string is terminated with the [ENTER] key, assuming the `minimum_allowed` parameter has been satisfied.

Each of the twelve imprinted keys on the keyboard represents several characters; the [ALPHA] key is used to scroll through each of the possible characters. The scroll sequence is inscribed on the key caps with two exceptions:

[0]: "0", "-" , " " (space), "+"
[#]: "#", "!", ":", "@", "=", "&"

The [BACKSPACE] key may be used to delete the last character entered.

The [CLEAR] key may be used to abort the read.

SVC_KEY_TXT() does not accept input from a function or screen-addressable key and will error beep on a key press from one of these keys.

Example

```
/* KEY_TXT.C */

#include <stdio.h>
#include <txosvc.h>

char dest_buff[20];
main() {

   printf("\fENTER SECRET NUM");

   /* An '*' will be displayed for each digit. */
   SVC_KEY_TXT(dest_buff, 2, 6, 1);

   /* Null terminate the string and display. */
   dest_buff[dest_buff[0]]=0;
   printf("\fNUMBER=%s",&dest_buff[1]);
}
```

SVC_MOD_CK()

Use SVC_MOD_CK() to validate or generate the Luhn check-digit for a counted string of ASCII digits representing an account number.

Prototype

```
#include <txosvc.h>
int SVC_MOD_CK( acc_num );
char *acc_num;
```

Parameters

SVC_MOD_CK() performs two Luhn checkdigit computations using the values of the ASCII characters in acc_num[] and combines the results to form its return code.

Returns

digit — The computation yields an ASCII digit ["0"-"9"], which is placed in the lower byte of the returned value. This value can be appended to acc_num[].

Notes

The first computation uses the formula for validating acc_num[], assuming that the last digit is the check digit. The computation yields 0xFF if an error is detected, and 0x00 otherwise; this value is placed in the upper byte of the returned value. The second computation uses the formula for generating the checkdigit for acc_num[], assuming that it doesn't contain one.

If the account number contains non-numeric characters, then:

1) The high-order byte will contain the number of digits not checked (0x01...0xFE).
2) The low-order byte will contain the non-numeric character encountered.

For example, suppose the caller passes the string "12C456". The result is 0x0443.

Example

See following page.

Example

```
/* MOD_CK.C */

#include <stdio.h>

char        valid_account[] = {"\0164818999999999 "},
          invalid_account[] = {"\0164818999999998 "},
    alpha_invalid_account[] = {"\016481899999A998 "};
int status;
main() {

   /* Evaluate what should be a valid account. */
   printf("\f%d  (0 IF VALID)",SVC_MOD_CK(valid_account)>>8);
   getchar();

   /* Evaluate what should be an invalid account. */
   printf("\f%d  (0 IF VALID)",SVC_MOD_CK(invalid_account)>>8);
   getchar();

   /* Extract the lower byte of the returned status from the      */
   /* invalid account number and concat it to the invalid number  */
   /* which should turn it into a valid account number.           */
   invalid_account[14] = SVC_MOD_CK(invalid_account) & 0x00FF;
   invalid_account[0] += 1; /* Increase count for counted string. */
   printf("\f%d  (0 IF VALID)",SVC_MOD_CK(invalid_account)>>8);
   getchar();

   /* If an alpha is in account string, the low-order byte contains  */
   /* the first non-numeric char found.  The high-order byte reports */
   /* the number of trailing chars (including the bad one reported)  */
   /* in the account string.                                         */
   status = SVC_MOD_CK(alpha_invalid_account);
   printf("\f'%c' IN ACCT STR",status&0x00FF);
   getchar();
   printf("\f%d CHRS NOT CHKED",status>>8);
}
```

SVC_PACK4()

Compresses buffers when writing to memory-based files.

Prototype

```
#include <txosvc.h>
unsigned int SVC_PACK4( dest, source, size );
char *dest, *source;
unsigned int size;
```

Parameters

`dest` is the address of a buffer to receive the compressed data (must be large enough to hold up to `size` bytes).

`source` is the address of a buffer containing data to be compressed.

`size` is the number of bytes in the source buffer to be compressed (must be less than or equal to 255).

Returns

`bytes_placed`	The number of bytes of output placed in the `dest` buffer.
`-1`	Failure. Check `size` parameter.

Notes

This routine may be used to compress buffers in a manner similar to that performed by `write_clvr()`. It is intended for ASCII text that is primarily numeric.

❖ *WARNING: Any data passed to this routine outside the range 0x00–0x5F is altered by the compression mechanism.*

Example

See following page.

Example

```
/* PACK4.C */

#include <stdio.h>
#include <strings.h>
#include <txosvc.h>

char src[20],dest[20],result[20];
main() {

   /* Build an alphanumeric string */
   strcpy(src, "1234567890ABCDEF");

   /* Pack the string, unpack it, then print it. */
   SVC_PACK4(dest, src, strlen(src));
   result[SVC_UNPK4(result, dest, strlen(dest))] = 0;
   printf("\f%s",result);
}
```

SVC_RAM_SIZE()

Returns the amount of static RAM in the terminal.

Prototype

```
#include <txosvc.h>
int SVC_RAM_SIZE( void );
```

Parameters None

Returns This call returns, in kilobytes, the amount of static RAM currently installed in the terminal.

Related Same as SVC_INFO_PLATFORM().

Example

```
/* RAM_SIZE.C */

#include <stdio.h>

main() {
   /* Display RAM size */
   printf("\f%dK BYTES RAM",SVC_RAM_SIZE());
}
```

SVC_RESTART()

Terminates execution of the current code file and starts executing a new code file.

Prototype

```
#include <txosvc.h>
int SVC_RESTART ( filename );
char *filename;
```

Parameters

`filename` is a null-terminated string containing the name of the new executable file to be executed.

Returns

If successful, the call never returns and the new code file begins executing.

`-1` if `filename` does not exist in the system.

Notes

If a null filename is specified as the parameter, the terminal will start the application specified by the *GO parameter in CONFIG.SYS, if defined. If *GO is not defined, that codefile which was first downloaded into the terminal is started.

To restart the currently running executable file, use the command: `SVC_RESTART(argv[0]);`

See *The C Programming Language, Second Edition, Kernighan and Ritchie, Section 5.10, Command Line Arguments* for information on passing executable file names as command line arguments.

This function is supported on EPROM Version 7 and higher.

Example

See following page.

Assume the following three programs are built and downloaded using:
DL CHAIN1.OUT CHAIN2.OUT CHAIN3.OUT *GO=""

```
/* CHAIN1.C */

#include <stdio.h>
#include <txosvc.h>

main(argc,argv)   int argc; char *argv[];    {

   /* Display the name of the currently executing application. */
   printf("\fRUN %s", argv[argc-1]);

   /* Start CHAIN2.OUT */
   SVC_RESTART("CHAIN2.OUT");
}

/* CHAIN2.C */

#include <stdio.h>
#include <txostd.h>
#include <txosvc.h>

main(argc,argv)   int argc; char *argv[];    {

   /* Display the name of the currently executing application. */
   printf("\fRUN %s", argv[argc-1]);

   /* Set *GO=CHAIN3.OUT */
   put_env("*GO", "CHAIN3.OUT", 10);

   /* System will restart application that *GO is set to. */
   SVC_RESTART("");
}

/* CHAIN3.C */

#include <stdio.h>
#include <txostd.h>
#include <txosvc.h>

main(argc,argv)   int argc; char *argv[];    {
```

Example continued on the following page.

```
    /* Display the name of the currently executing application. */
    printf("\fRUN %s", argv[argc-1]);

    /* Delete the *GO entry from CONFIG.SYS */
    put_env("*GO", "", 0);

    /* Restart the oldest code file (CHAIN1.OUT). */
    SVC_RESTART("");
}
```

Starting the terminal's application will result in the following display:

```
RUN CHAIN1.OUT
RUN CHAIN2.OUT
RUN CHAIN3.OUT
RUN CHAIN1.OUT
```

SVC_STORE()

Allows the terminal user access to the system mode to change or create a keyed file.

Prototype

```
#include <txosvc.h>
int SVC_STORE( filename );
char *filename;
```

Parameters `filename` is a null-terminated string naming the new or existing keyed file to edit. As shown in the example code on the following page, the file name is not case sensitive.

Returns

0	Success.
-1	Failure.

Notes `SVC_STORE()` implicitly opens and closes the keyed file.

Calling this routine brings up the "RECALL?" prompt on the terminal display. The terminal user is then able to scroll through all items in the specified keyed file; changing, inserting and deleting entries in a normal editing fashion. The user presses the [CLEAR] key upon completion. Any changes to the file are saved, and the application resumes at the instruction following the `SVC_STORE()` request.

Note This function is supported on EPROM Version 10 and higher.

Example See following page.