SAE International™

SURFACE VEHICLE RECOMMENDED PRACTICE

SAE J1683

CAN-CELLED JUN2004

Issued 1997-02
Cancelled 2004-06

Superseding J1683 FEB1997

MS-DOS™ Interface for SAE J1708 Communications

Foreword—The following are the perceived needs for the communication driver.

a. Communicates with all supplier software packages. This requires a well-defined and well-behaved interface.
b. Allows hardware adapters to be transparent to the application programmer: no portion of the interface to the application program should be tied to any specific method or hardware configuration.
c. Allows OEMs to differentiate their software product offerings on a proprietary basis while still maintaining consistency between vendors.
d. Supports the SAE J1708/SAE J1587 network protocol for both hardware and software.
e. Maintains software language independence. This means a non-linkable driver that is register-based for its communications interface. No portion of the interface should require specific library or language support.
f. Stays resident while application programs are activated and de-activated. This allows one vendor's software to activate another's while each maintains its own security and proprietary structure.
g. When other operating systems are being run while attempting to utilize this driver, all rules relating to a DOS based TSR must be followed. Example: The TSR should not be loaded from a DOS window in MS-Windows™.
h. Maintains compatibility with MS-DOS™ operating system interface procedures and with current and future DOS versions so it won't be designed out of the hardware base.
i. Operates efficiently, in order to allow low-end machines (such as a 286 based PC) to run available software.
j. Allows data to be transferred to global-memory buffers as requested by application programs. Additionally, supplies a filtering mechanism for these buffers in order to efficiently route data streams to modules that desire them. This maintains the independence of one software module from the next while minimizing the programmer's efforts.
k. Implements a timer-triggered interrupt-driven system for transferring data over the networks.

TO PLACE A DOCUMENT ORDER:
Tel: 877-606-7323 (inside USA and Canada)
Tel: 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: custsvc@sae.org
SAE WEB ADDRESS: http://www.sae.org

TABLE OF CONTENTS

1. ***Scope***—The communication driver described in this SAE Recommended Practice resides on the PC between the SAE J1708 hardware and the individual application programs developed by each supplier. From the perspective of the application program, the SAE J1708 hardware is totally transparent. This means diagnostic programs can operate on any PC using any SAE J1708 adapter hardware. The diagnostic (application) program sees no difference from one hardware base to the next. All hardware-specific tasks are handled by this document. This allows for complete portability of software programs. It also guarantees complete functionality from one hardware platform to the next.

The standard presented applies to computer systems that use the MS-DOS operating system. Design standards that contain the same functionality may be defined in the future for other operating systems.

1.1 **Purpose**—A mechanic's familiarity with diagnostics and the long-term effectiveness of training will be improved by standards on original equipment manufacturers (OEM) and aftermarket service tools. This document provides a programming standard for a communication driver, hereafter referred to as an Application Program Interface (API), used to develop computerized diagnostic aids.

2. ***References***

2.1 **Applicable Publications**—The following publications form a part of the specification to the extent specified herein. Unless otherwise indicated the latest revision of SAE publications shall apply.

2.1.1 SAE PUBLICATIONS—Available from SAE, 400 Commonwealth Drive, Warrendale, PA 15096-0001.

SAE J1587—Electronic Data Interchange Between Microcomputer Systems in Heavy-Duty Vehicle Applications

SAE J1708—Serial Data Communication Between Microcomputer Systems in Heavy-Duty Vehicle Applications

3. ***Guidelines***—One major design standard needed, which will provide an environment for developers of diagnostic software, is a communication interface standard between the PC and SAE J1708 data link. Many hardware devices can be used to connect the PC to the SAE J1708 data link in a truck. Three of the current methods that suppliers have used to connect the PC to the SAE J1708 data link are as follows:

a. Hand-held tool which connects to the RS-232 I/O port on the PC and to the SAE J1708 data link
b. Interface box between the RS-232 on the PC and the SAE J1708 data link
c. PC interface card that plugs into a PC bus slot and has a cable which connects to the SAE J1708 data link

To avoid the problem of developers creating multiple programs for each hardware scheme (which increases cost and confusion to the end user) and to provide a standard programming environment for suppliers, a standardized interface for a communication driver has been derived.

The communication driver is a program which contains the interface to the particular hardware apparatus and application program. This driver is referred to as an Application Program Interface (API).

The interface for the communications driver (API) consists of system-level calls to the SAE J1708 functions via the use of registers and interrupts. The driver would be written in assembly language (or possibly in the "C" language) to allow for greater speed.

Since application programmers typically use a high-level language, an example library of application program calls has been included. Note, the library routines are language specific and must be compiled with the user application program. However, the standardized communication driver is a stand-alone entity and only requires that it be loaded upon power-up.

User programs can interface to the SAE J1708 functions via the system-level calls in the communication driver (API) or through the provided application program calls.

The communication driver (API) consists of two major software sections: the SAE J1708 communication functions and the 1 ms timer interrupt functions. Although the main purpose of this document is to supply SAE J1708 communication functions, the timer interrupt functions are required to start and pace transmissions and receptions. To accommodate communication functions, a 1 ms timer resolution is required.

Since it would be advantageous for most application programs to have access to timer functions, this API allows access to them.

The timer functions that are accessible by the application programmer are based on a 10 ms user resolution. If a DOS timer (such as interrupt 8) is used, then all DOS functions relating to that timer must be maintained. No disruption of any DOS functionality is to occur.

The timer and serial interrupt handlers will be the same source module so that they can be incorporated in a Terminate and Stay Resident (TSR) program. Software interrupt "27H" is one method to terminate execution of the program but reserve part or all of its memory so that it will not be overlaid by the next transient program to be loaded. The user program will be loaded into the remaining memory. The memory utilized by the resident portion of this API should be about 12 KB in size.

Timers are accessed by causing a software interrupt with a function value (register "AH") set to 0. The sub-function value (register "AL") indicates what action is being requested (see Table 1).

TABLE 1—TIMER SETUP COMMANDS (REGISTER "AH" SET TO 0)

Sub-Function Value	Description
0	Initialize timer
1	Reset timer
2	Restore timer to DOS
3	Get timer information
4	Install general purpose timer
5	Remove general purpose timer
6	Install interrupt timer
7	Remove interrupt timer
8	Restart timer
9-50	Reserved
51-255	User definable

The SAE J1708 bus is accessed in the same manner using the same interrupt as the timers. Its function value (register "AH") is set to 1. The sub-function value is passed via register "AL" (see Table 2), as with the timer sub-function.

TABLE 2—SAE J1708 COMMUNICATION SETUP COMMANDS (REGISTER "AH" SET TO 1)

Sub-Function Value (register "AL")	Description
0	Initialize SAE J1708 channel
1	Reset error conditions
2	Restore DOS handle
3	Get SAE J1708 driver information
4	Install usage buffer
5	Remove usage buffer
6	Install MID/PID receive filter
7	Remove MID/PID receive filter
8	Install transmit broadcast buffer
9	Remove transmit broadcast buffer
10	Transmit message packet
11	Transfer buffer immediately
12	Install receive error buffer
13	Remove receive error buffer
14	Install timed transmit buffer
15	Restore timed transmit buffer
16	Allocate TSR buffers
17	De-allocate TSR memory pool
18	Reset communications network timer
19	Hardware test
20	Install performance buffer
21	Remove performance buffer
22-50	Reserved
51-255	User definable

3.1 Return Status—The communication driver returns a status or pass/fail result of the requested operation. It does this by returning a binary value in register "AL." Zero (0) indicates that all went well and the task is accomplished. Non-zero indicates an error has occurred. The errors that may occur are defined in Table 3.

TABLE 3—RETURN STATUS OF TASKS

Register "AL"	Description
0	All went well, successful
1	Function code is out of range
2	Sub-function code is out of range
3	Request is not supported by this driver
4	Insufficient room in buffer
5	ID value was not found in active list
6	Interrupt is not available for timer use
7	Requested parameter does not exist
8	Request parameter was assigned
9	Parameter list is invalid
10	Request for action has failed. Failure not determined
11-50	Reserved
51-225	User definable

3.2 Interrupt Selection—This section addresses some concerns relating to the Application Program interface (API) interrupt number (or vector). MS-DOS does not have any accommodations for registering vectors for specific purposes or to specific organizations. MS-DOS does, however, classify a set of vectors as user-definable: 60 Hex through 7F Hex. These vectors are open for anyone to use in an application, so they are available for uses such as this communication driver. However, they are also available for other applications, such as networks, I/O devices, etc.

This leaves the problem of identifying the API vector to be sure it is installed and at the interrupt interface vector that the application program is assuming. In addition, it must be assumed that other software packages may select the same interrupt level that this TSR has defined as its own. It might be desirable to allow some of these software packages to be installed or to run concurrently with this TSR. It would, therefore, be advantageous to have the ability to reassign the vector level to another opening.

Interrupt 62 Hex is designated as the default entry vector level. To install this TSR's API entry point at a level other than 62 Hex and to communicate the new location to application programs, the new vector assignment is passed into the transient portion of the TSR via its environment. All values being passed via the environment are to be preceded by an identifying ASCII string. For the API interrupt level, the string "INTR=" is to be used. For a serial port controlled adapter, the port selection identifier is the string "COMM=." The transient module of the TSR should check the environment for any parameters. If present, the parameters should be compared to the identifying strings to see which parameters are to be altered. Once determined, the TSR should take the appropriate action. This string method allows mixed-order or missing parameters. Thus, only the changes being requested, if any, need to be passed to the program.

As an example, assume the TSR's executable is named DRIVER.exe and that it communicates through a serial port. To install the TSR, you merely type "DRIVER<Return>." The TSR will be installed with its default configuration. If you wish to change its serial port to port 2, type "DRIVER COMM=2<Return>." To alter the API interrupt number to 6E Hex, type "DRIVER INTR=6E<Return>." Any order or combination can be typed in.

To identify the TSR as being present or not, a 4-byte code immediately preceding the TSR's API entry point is to be coded into the resident portion of the TSR. This allows application programs to search the vector table to locate the level at which the TSR resides. The following is an example of this recommendation and how it can be implemented.

3.3 Examples—First is the entry point to the TSR itself. The segment and offset of the label "TSR_Entry" is our vector. It is loaded into the vector table.

```
                                ;
ID_Label:                       ;
NOP                             ; This section allows application programs to
NOP                             ; identify this TSR by accessing the vector table
NOP                             ; addresses.  Pattern c5 3a a3 5c


TSR_Entry:                      ; Entry point for mixed-language use
                                ;
PUSH BP                         ; Save base pointer,
PUSH DS                         ; the user's data segment,
PUSH BX                         ; and register BX.
MOV BP,SP                       ; Load stack position into base pointer.
                                ;
```

Following is a section of the transient code that installs the interrupt in the vector table and then loads the identifier code, 32 bits, at the label immediately before the vector address.

```
                                        ;
MOV AL,API_Intr_Level                   ; Load vector number again.
MOV AH,25H                              ; Function 25H: install interrupt vector.
MOV DX, OffSet Resident_Group:TSR_Entry
                                        ; Load offset of TSR's API.
MOV BX,Resident_Group                   ; Load segment of new handler.
MOV DS,BX                               ; Put it into data segment.
INT 21H                                 ; Install the interrupt vector.
                                        ;
MOV AX,Resident_Group                   ; Initialize DS to the resident program's code
MOV DS,AX                               ; segment.
MOV BX,OffSet ID_Label                  ; Load the address of 4 bytes before 62H vector.
MOV Byte Ptr[BX],0C5H                   ; Put in the bit pattern expected.
INC BX                                  ; Bump to second byte of group.
MOV Byte Ptr[BX],03AH                   ; Load second byte value.
INC BX                                  ; Bump to third byte of group.
MOV Byte Ptr [BX],0A3H                  ; Load third byte value.
INC BX                                  ; Bump to fourth byte of group.
MOV Byte Ptr[BX],05CH                   ; Load fourth byte value.
                                        ‘
```

Following are two examples of code that could be used in the application program to find and identify the location of the TSR.

```
#define                 TSR_ID          0x5ca33ac5
unsigned        char    Found_TSR:
unsigned        long    far *Vector_Address
unsigned        long    far *ID_Number;
Found_TSR = 0;
Vector_Index = 0x60;
Vector_Address = (unsigned long far *)0X00000180; while (( Vector_Index < = 0x7f)
&& (!Found_TSR ))
{
ID_Number = (unsigned long far *)(*Vector_Address);
ID_Number–;
                if (TSR_ID = = *ID_Number)
Found_TSR = Vector_Index;
Vector_Address ++;
Vector_Index ++;
}
                                        ;
TSR_ID_Lower EQU A35CH                  ; offset of first vector to check.
TSR_ID_Upper EQU C53AH                  ; offset of last vector in check list.
Lower_Vector EQU 0180H                  ; offset of first vector to check.
Upper_Vector EQU 0200H                  ; offset of vector search end.
                                        ;
Find_TSR:                               ;

XOR AX,AX; Clear AX to set to vector table segment.
MOV ES,AX                               ; and use AL as the vector identifier at.
MOV DI,Lower_Vector                     ; end.  Load low end of vector table search offset.
```

```
Search_Loop:                    ;
MOV BX,ES:[DI+2]                ; Load the segment of the vector from table.
MOV DS,BX                       ; Put it into the data segment.
MOV BX,ES:[DI]                  ; Load the offset of the vector.
SUB BX,2                        ; Drop offset back to upper 2 bytes if ID.
CMP Word Ptr[BX],TSR_ID_Upper   ; Compare to the upper word of ID code.
JNZ Not_TSR                     ; If no match, then branch to look at next vector.
                                ;
SUB BX,2                        ; Drop offset back to lower 2 bytes of ID.
CMP Word Ptr[BX],TSR_ID_Lower   ; Compare to the lower word of ID code.
JNZ Not_TSR                     ; If no match, then branch to look at next vector.
                                ;
MOV AX,DI                       ; Load the vector table offset into AX.
SHR AX,1                        ; Divide this offset by 4 to obtain the
SHR AX,1                        ; interrupt number of the API.  It's in AL.
RET                             ; Return with AL equal to API interrupt.
                                ;
Not_TSR:                        ;
ADD DI,4                        ; Bump vector table offset to next vector.
CMP DI,Upper_Vector             ; See if we are at the end of our search.
JC Search_Loop                  ; If not, then, loop back up to check this one.
                                ;
RET                             ; Otherwise, it's not there. Return with AL = 0
                                ;
```

3.4 Application Program Clarification—Throughout this document, registers, memory location, and their individual bits (elements) are defined as part of the interface specification. Those elements that are not specifically assigned are classified as either "Reserved" or "User Definable."

Elements that are labeled "**Reserved**" are not to be used by the programmer of either the API Driver (this document) itself or the applications. Reserved elements are left for future assignment by SAE and should always be cleared or set to zero to maintain future compatibility.

Elements that are labeled **"User-Definable"** are left open and can be used for whatever purpose is deemed appropriate by the individual programmer. User-Definable elements are left to the discretion of the developer of the API driver. Whatever application, beyond this document, is developed utilizing these "User-Definable" elements must appear transparent to any and all application software packages, so long as the application software package clears all of the "User-Definable" elements.

4. Application Program Interface for Timer Interrupt Functions

4.1 Initialize Timer—This function initializes only the timer interrupt portion of the communication driver to produce a 10-ms timer tick. To do so, it initializes all timer-related hardware—setting up any vectors needed—and it saves all information relating to DOS that it will alter so that it may be restored later.

Call with:

AH = 0 = Function: Timer
AL = 0 = Sub-Function: Initialization
BX = User Definable

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function value out of range
8 = Pre-assign, initialization done

Coding Examples:

```
Assembly Language:                    ;
    MOV AX,0000H                      ; Function/Sub-function = Timer Initialization
    MOV BX,0000H                      ; Zero user definable register
    INT 62H                           ; Software gateway to TSR
    ;    Error Checking here          ;
```

C Language:

```
/*  include C library for REGS definition and define TSR interrupt level        */
    #include  <dos.h>
    #define TSR_Intr 0x62
    union REGS inregs, outregs;
    {
    inregs.x.ax = 0x0000;             /* Function/Sub-function = Timer initialize  */
    inregs.x.bx = 0x0000;             /* Zero, user definable register
    int86(TSR_Intr, &inregs, &outregs);  /* Cause interrupt to access TSR          */
    /* Error checking here                                                        *\
    }
```

NOTE— This function is not needed if the communications section of the TSR is being used. It handles initialization of the timer interrupt ONLY. This function and sub-function number is reserved for compatibility: specifically, for stand-alone applications when no communications module is required.

4.2 Reset Timers—This function is a software reset of the timer interrupt handler. It resets vectors, memory, etc., and purges all currently installed timers from the activity list.

Call with:

AH = 0 = Function: Timer
AL = 1 = Sub-Function: Reset

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function value out of range
7 = No-exist, timer has not been initialized

Coding Examples:

```
Assembly Language:                    ;
    MOV AX,00001H                     ; Function/Sub-function = Timer Reset
    INT 62H                           ; Software gateway to TSR
    ;    Error Checking here          ;
```

C Language:

```
/*
 *   include C libraries for REGS definition and define TSR interrupt level.
*/
     #include <dos.h>
     #defineTSR_Intr 0x62
     union REGS inregs, outregs;
     {
     inregs.x.ax = 0x0001;                /* Function/Sub-function = Timer Reset,    */
     int86(TSR_Intr, &inregs, &outregs);  /* Cause interrupt to  TSR, Gateway        */
     /* Error checking here               */
     }
```

NOTE—This reset purges all timers that have been installed. It should be noted that the reset function under the communications section does a soft re-initialization of the communication port and does not effect the timers. It is used as an error condition recovery. This reset, however, is for the purpose of clearing a timer, rather than error recovery of any kind.

4.3 Restore DOS—This function restores the interrupt, clears memory, and returns timer control to DOS. It restores to pre-initialization condition, but does not eliminate the resident program. However, it makes the timer program dormant so that it requires re-initialization to become functional again. Timer restore will only work if timer initialization was used. If the communicator initialization was used, an error status of 7 is returned.

Call with:

AH = 0 = Function: Timer
AL = 2 = Sub-Function: Restore to DOS
BX = User Definable

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function value out of range
7 = No restore buffer available, has not been initialized

Coding Examples:

```
     Assembly Language:                ;
     MOV AX,0002H                      ; Function/Sub-function = DOD Restore
     MOV BX,0000H                      ;  Zero user definable register
     INT 62H                           ; Software gateway to TSR
;    Error Checking here               ;
```

C Language:

```
/*
 *include C libraries for REGS definition and define TSR interrupt level.
*/
    #include <dos.h>
    #define     TSR_Intr 0x62
    union REGS inregs, outregs;
    {
    inregs.x.ax = 0x0002;                   /* Function/Sub-function = DOS Restore */
    inregs.x.bx = 0x0000;                   /* Zero, user definable register       */
    int86(TSR_Intr, &inregs, &outregs);     /* Cause interrupt to  TSR, Gateway    */
    /*      Error checking here                                                    */
    }
```

NOTE—As with the timer initialize command, this restore-to-DOS command is not used when the communications portion of the TSR is active. It is required only for stand-alone use of the timer interrupt.

4.4 Get Timer Information—To determine the usability of certain information relating to the TSR itself. This module returns information relating to the manufacturer, hardware used, SAE standards compatibility, version number, etc.

Call with:

AH = 0 = Function: Timer
AL = 3 = Sub-Function: Get Timer Version and Capability Information
CX =Size in bytes of buffer area pointed to be ES:DI
ES:DI==>Segment and Offset where information is to be stored

Returns with:

AL = 0 = Status OK
1 = Function value is out of range
2 = Sub-function is out of range
3 = This function is not supported by TSR
4 = Insufficient room in user's buffet

Coding Examples:

Assembly Language:

```
                                ;
Timer_Info      DB 64 Dup(?)    ; Buffet area for TSR information storage
Info_Size       EQU $-timer_Info ; Size, in bytes, of buffer area
                                ;
MOV     AX, Seg Timer_Info      ; Load segment of information buffer
MOV     ES,AX                   ; Move it into extra segment register
MOV     DI, Offset Timer_Info   ; Load offset of buffer address into destination index
MOV     CX,Info_Size            ; Load size of buffer into register pair CX
MOV     AX,0003H                ; Function/Sub-function = Get Timer Information
INT     62H                     ; Software gateway to TSR
;     Error Checking here       ;
```

C Language:

```
*/
 *include C libraries for REGS definition and define TSR interrupt level.
*/
     #include <dos.h>
     #define  TSR_Intr 0x62
     union REGS inregs, outregs;
     struct SREGS segregs:
/*
*    Define a structure with the elements that are to be returned in the information block.
*    The items in the structure below are compatible with Version 1.00 of this proposal.
*    Versions beyond this may use a union to hardware in definitions to accommodate a variety
*    of later versions.
/*
     struct Timer_Information
     }
     unsigned int    Buffer_Size              /*  Valid data byte count                */
                     Manufacture_Version,     /*  Manufacture specific                 */
                     SAE_Version;             /*  SAE standard supported               */
     char            Author_ID[5],            /*  TSR software developer               */
                     Hardware[3];             /*  Hardware supported                   */
     unsigned int    DOS_Version,             /*  MS-DOS version                       */
                     Timers_Total,            /*  Timer capacity                       */
                     Timer_Vacancy,           /*  Timers still available               */
                     Interrupts_Vacancy,      /*  Interrupts available                 */
                     Reserved [15];           /*  For later expansion                  */
     ;
     struct Timer_Info Timer_Infor;           /*  Load segment of information buffer   */
     struct Timer_Info  far  *Info_Ptr;       /*  Move it into extra segment register  */
     {
     Info_Ptr     = &Timer_Info;
     segregs.es = FP_SEG (Info_Ptr);          /*  Move into extra segment register     */
     inregs.x.di = FP_OFF (Info_Ptr);         /*  Load offset of buffer address into   */
                                              /*  destination index                    */
     inregs.x.cx=sizeof (Timer_Info);         /*  Load size of buffer in register pair CX*/
     inregs.x.ax=0x0003;                      /*  Function/Sub-function = Timer Info   */
     int86x(TSR_Intr, &inregs,                /*                                       */
     &outregs, &segregs);                     /*  Cause interrupt to TSR               */
     /* Error checking here                   */
     }
```

NOTE—Segment (ES) and offset (DI) point to a buffer area (seeTable 4) in the application program's data area. Register CX contains a binary count of the size of this buffer area in bytes. It prevents advanced versions of the TSR from overwriting beyond buffer limits.

TABLE 4—INFORMATION BLOCK—TIMER SECTION

Offset	Label	Description
00H-01H	Buffer Size	Byte count of valid data in buffer area. Does not count itself.
02H-03H	TSR Version	TSR Program, Version number, manufacture specific, Initial Version 1.00. a.b.: a = byte 03H, b = byte 02H, binary.
04H-05H	SAE Version	Version of SAE standard fully supported. a.b.: a = byte 05H, b = byte 04H, binary.
06H-0AH	Author	Manufacture author of Program SAE: VMRS supplier codes (5 character ASCII)
0BH-0DH	Hardware	Hardware that the software is compatible with (3 characters ASCII)
0EH-0FH	DOS_Version	Minimum DOS version supported by the TSR program. a.b.: a = byte 0FH, b = byte 0EH, binary.
10H-11H	Timers Total	Total number of times that can be installed at any one time. This is dictated by RAM available.
12H-13H	Timers Installed	Number of timers currently installed.
14H-15H	Interrupts_Installed	Number of vectors currently in use.
16H	Vector Low	Interrupt vector number where search is started for a free interrupt in the vector table.
17H	Vector High	Interrupt vector number where search is terminated looking for available interrupt in vector table.
18H-3FH	Reserved	Reserved for future assignment by SAE.
40H-7FH		User definable.

4.5 Install Timer—Installs a 16-bit unsigned timer with a minimum interval time of 10 ms. Any multiple of the minimum interval time can be set. The timer can be set to increment or decrement continuously, or just once.

Call with:

AH = 0 = Function: Timer
AL = 4 = Sub-Function: Install Timer
BX = Control flag of timer
CX = Timer's reset or terminal value
ES:DI==> Segment and offset of timer

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function value out of range
4 = Buffer area full, no room remaining
7 = Timer not initialized

AL = Timer identification number

Coding Examples:

Assembly Language:

```
                                    ;
Timer Value     EQU 1000            ; Timer reset value = 10 seconds
                                    ; 1000 times timer resolution (10 ms)
User_Timer      DW (?)              ; Actual timer to be used.
User_ID         DB (?)              ; ID or reference number of timer.
                                    ;
MOV AX,Seg User_Timer               ; Load segment where timer resides
MOV ES, AX                          ; and puts it into the ES register.
MOV DI,Offset User_Timer            ; Load timer's address offset

MOV CX,Timer_Value                  ; Load timer's reset/terminal value.
MOV AX,0004H                        ; Set Function/Sub-function = Install Timer
INT 62H                             ; Do cause interrupt, install timer.
; Error checking here               ;
                                    ;
MOV User-ID, AH                     ; Save timer's reference number.
```

C Language:

```
*/
 *include C libraries for REGS definition and define TSR interrupt level.
*/
    #include <dos.h>
    #define TSR_Intr 0x62
    #define Timer_Value 1000
    #define OneShot 0x0002
    #define Increment 0x0001
    union REGS inregs, outregs;
    struct SREGS segregs:
    unsigned int User_Timer;
    unsigned char User_ID;
    unsigned int far *timer_ptr;*
    {
    timer_ptr  = &User_Timer;           /*  Load segment of timer location          */
    segregs.es = FP_SEG(timer_ptr);     /*  Load segment of timer location          */
    inregs.x.di= FP_OFF(timer_ptr);     /*  Load offset of timer address            */
    inregs.x.cx =Timer_Value;           /*  Load terminal timer value               */
    inregs.x.bx =OneShot + Increment;   /*  Incrementing oneshot timer              */
    inregs.x.ax = 0x0004;               /*  Function/Sub-function = Install Timer   */
    int86x(TSR_Intr, &inregs, &outregs, &segregs);
                                        /*  Cause interrupt to TSR                  */
    /* Error checking here                                                         */
    User_ID = outregs.h.ah;             /*  Save this timer's ID reference number   */
}
```

NOTE—

Control_Flag, BX:

Bit0: Increment/Decrement

If this bit is set, the timer will be incremented by the TSR. If incremented, then it will only count up to the value in CX (timer reset value). Otherwise, it will be decremented. If it is to be decremented, then it will count down to zero.

Bit1: Oneshot/Continuous
By setting this bit, the timer will count up or down to its terminal value (timer reset value) and stop, a oneshot. If this bit is cleared, then when the timer reaches its terminal count (0 for a decrementing timer and the timer reset value for an incrementing timer) it automatically resets itself and continues to count.
Bit2-Bit 7: Reserved for future SAE assignment
Bit8-Bit15: User definable

Timer Reset Value, CX: For a decrementing timer, this is the initialization and the restart value that is loaded by the TSR. If the timer is a continuous timer as well as a decrementing, then this value is loaded upon reaching zero. For an incrementing timer, this is the terminal count where it either stops or resets itself to zero.

Timer_SEG, ES: Segment where the unsigned 16-bit timer value resides.

Timer_Off, DI: Offset of timer.

Vector Assignment, CL: If this routine is successful in assigning a timer interrupt, it will return the interrupt number that was assigned.

The assignment of an interrupt is done by the TSR itself. It is reprogrammed with a high- and a low-vector number for "where to conduct a search." It starts at the low-vector position in the system's vector table and searches for a free, unused position. This search continues sequentially until the high-vector position is reached or an open-vector is found and assigned.

Care should be taken when writing one of these interrupt handlers. Since the timer routine itself will vector here via a software cause, interrupt time is limited.

4.6 Remove Timer—This function removes a 16-bit timer with a minimum interval time of 10 ms.

Call with:

AH = 0 = Function: Timer
AL = 5 = Sub-Function:
CL = Timer ID#

Returns with:

AL = 0 = Status OK
1 = Function number out of range
2 = Sub-function out of range
5 = If invalid ID number
7 = Not exist, not initialized, or timer not active

Coding Examples:

Assembly Language:

```
                                    ;
User_Timer      DW (?)              ; Timer itself
User_ID         (DB(?)              ; ID or TSR reference number of timer
                                    ;
MOV CL,User_ID                      ; Load timer's reference number
MOV AX,0005H                        ;  Function/Sub-function = Remove Timer
INT 62H                             ; Software gateway to TSR
;     Error Checking here           ;
```

C Language:

```
*/
 * include C libraries for REGS definition and define TSR interrupt level.
*/
#include <dos.h>
#defineTSR_Intr 0x62
union REGS inregs, outregs;
unsigned int User_Timer
unsigned char User_ID;
{
inregs.h.cl= User_ID;                /*  Load timer's ID number                */
inregs.x.ax= 0x0005;                 /*  Function/Sub-function = Remove Timer  */
int86x(TSR_Intr, &inregs, &outregs); /*  Cause interrupt to TSR                */
/* Error checking here               */
}
```

NOTE—**Timer_ID#, CL:** This is a binary value that identifies which timer is being referenced. It is the value assigned and returned by Install_Timer," Function 0, Sub-function 4. By passing in a value of 255 (0FF Hex) in this register, all timers will be removed.

Since the two sub-functions "Install_Timer (4)" and "Install_Timer_Interupt (6)" are functionally very similar, they can be saved by the TSR in the same buffer area. Thus, ID numbers returned by these two functions are unique: if "Install_Timer" assigns an ID of 2, then "Install_Timer_Interrupt" cannot use that ID number until it is released by "Remove_Timer." In addition, even though these two functions work out of the same buffer, their corresponding remove functions (5 and 7) can only remove timers that have been assigned by their own install functions.

4.7 Install Timer Interrupt—This function installs a timer interrupt which vectors to the user program. The minimum interval times is 10 ms. Any multiple of the minimum interval time can be set. The timer can be set to increment or decrement. It can be continuous or a oneshot. When the timer reaches its terminal value, it causes a software interrupt to the code pointed to by DS:DX.

Call with:

```
AH =   0   =   Function: Timer Interrupt
AL =   6   =   Sub-Function: Install Timer Interrupt
BX =           Control flag for timer
CX =           Timer interval (10 ms per bit)
DS:DX  ==>     Segment and offset of the interrupt handler in user program
ES:DI  ==>     Pointer to timer location
```

Returns with:

AL =	0 =	Status OK
	1 =	Function number out of range
	2 =	Sub-function out of range
	4 =	Buffer area full, no timer room
	6 =	No interrupt is available
	7 =	Not initialized
AH =		Contain Timer ID#
CL =		Vector assignment

Coding Examples:

Assembly Language:

```
  TSR_Intr        EQU 062H          ; Gateway interrupt number
  Timer_Value     EQU 1000          ; Timer reset value = 10 s
  OneShot         EQU 0002H         ; Oneshot timer operation
  Increment       EQU 001H          ; Increment timer
                                    ;
  User_Timer      DW(?)             ; The timer itself
  User_ID         (DB(?)            ; Storage for timer's reference number
                                    ;

    MOV AX,Seg User_Timer           ; Load segment of the timer itself
    MOV ES,AX                       ; Move it into extra segment register
    MOV DI,Offset User_Timer        ; Load offset of timer's address into destination
                                    ; index
    MOV BX,OneShot                  ; Load timer configuration, decrement/oneshot
    MOV CX,Timer_Value              ; Load timer configuration flag
    MOV AX,Seg Timer_Intr           ; Load segment of the interrupt routine
    MOV DS,AX                       ; Move it into extra segment register
                                    ; index
    MOV AX,006H                     ; Function/Sub-function = Install timer interrupt
    INT TSR_Intr                    ; Software gateway to TSR
    ;    Error checking here        ;
                                    ;
    MOV User_ID, AH                 ; Save timer's ID number
                                    ;
;_____________________________;
                                    ;
    Timer_Intr:                     ; This is the interrupt handler that is vectored to
    IRET                            ; by the timer routine.  Interrupt code goes here
```

C Language:

```
/*  include C library for REGS definition and define TSR interrupt level.*/
    #include <dos.h>
    #define TSR_Intr          0x62
    #define Timer_Value       1000
    #define OneShot           0x0002
    #define Increment         0x0001
    union REGS inregs, outregs;
    struct SREGS segregs:
    interrupt far User_intr( );
    unsigned int User_Timer;
    unsigned char User_ID;
```

```
    unsigned int far  *timer_ptr;*
    {
    timer_ptr =&User_Timer;                 /*  Load timer's address into a far pointer    */
    segregs.es = FP_SEG (timer_ptr);        /*  Put segment in extra segment register      */
    inregs.x.di  = FP_OFF (timer_ptr);      /*  Load offset of timer address               */
    segregs.ds = FP_SEG (User_Intr);        /*  Load segment of interrupt routine          */
    inregs.x.dx = FP_OFF (User_Intr);       /*  Load offset of interrupt address           */
    inregs.x.cx = Timer_Value;              /*  Load terminal timer value                  */
    inregs.x.bx = OneShot + increments;     /*  Incrementing oneshot timer                 */
    inregs.x.ax = 0x0006;                   /*  Function/Sub-function = Install Interrupt  */
    int86x(TSR_Intr, &inregs, &outregs, &segregs);                                         */
                                            /*  Cause interrupt to TSR                     */
    /* Error checking here                  */
    User_ID = outregs.h.ah;                 /*  Save this timer's ID reference number      */
    }
    interrupt far control_break()
    {
    *** code for interrupt goes here ***
}
```

NOTE—

Control _Flag, BX:

Bit0:Increment/Decrement

If this bit is set, the timer will be incremented by the TSR. If incremented, then it will only count up to the value in CX (timer reset value). Otherwise, it will be decremented. If it is to be decremented, then it will count down to zero.

Bit1:Oneshot/Continuous

By setting this bit, the timer will count up or down to its terminal value (timer reset value) and stop, a oneshot. If this bit is cleared, then when the timer reaches its terminal count (0 for a decrementing timer and the timer reset value for an incrementing timer) it automatically resets itself and continues to count.

Bit2-Bit7: Reserved for future SAE assignment

Bit8-Bit15:User definable

Timer Reset Value, CX:

For a decrementing timer, this is the initialization and the restart value that is loaded by the TSR. If the timer is a continuous timer, as well as decrementing, then this value is loaded upon reaching zero. For an incrementing timer, this is the terminal count where it either stops or resets itself to zero.

Timer_Seg, ES:

Segment where the unsigned 16-bit timer value resides.

Timer_Off, DI:

Offset of timer.

Interrupt Handler Segment, DS:

This is the segment address for the routine in user's code where we should vector to. It is loaded into the vector table.

Interrupt Handler Offset, DX:

Offset address of user's interrupt handler.

Vector Assignment, CL:

If this routine is successful in assigning a timer interrupt, it will return the interrupt number that was assigned.

The assignment of an interrupt is done by the TSR itself. It is reprogrammed with a high- and a low-vector number for "where to conduct a search." It starts at the low-vector position in the system's vector table and searches for a free, unused position. This search continues sequentially until the high-vector position is reached or an open vector is found and assigned.

Care should be taken when writing one of these interrupt handlers. Since the timer routine itself will vector here via a software cause, interrupt time is limited.

4.8 Remove Timer Interrupt—Removes the timer interrupt which vectors to the user program. This function removes both the timer and the vector.

Call with:

AH = 0 = Function: Timer
AL = 7 = Sub-Function:
CL = Timer ID#

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function out of range
5 = Invalid ID number
7 = Not initialized or ID does not exist

Coding Examples:

Assembly Language:

```
User_ID         (DB(?)                  ;
                                        ; ID or TSR reference number of timer
                                        ;

MOV   CL,User_ID                        ; Load timer's reference number
MOV   AX,007H                           ; Function/Sub-function = Remove interrupt
INT   62H                               ; Software gateway to TSR
                                        ;
;     Error checking here               ;
```

C Language:

```
/*
 *Include the C libraries for REGS definition.   Define interrupt level.
/*
     #include   <dos.h>
     #define TSR_Intr      0x62
     union REGS inregs, outregs;
     unsigned char User_ID;
     {
     inregs.h. cl  = User_ID;                  /*   Load timer's ID number                    */
     inregs.x.ax = 0x0007;                     /*   Function/Sub-function = Remove Interrupt*/
     int86x(TSR_Intr, &inregs, &outregs);                                                      */
                                               /*   Cause interrupt to TSR                    */
     /*   Error checking here                                                                  */
     }
```

NOTE—**Timer_ID#, CL:** Is a binary value that identifies which timer is being referenced. It is the value assigned and returned by "Install_Timer," Function 0, Sub-function 4. By passing in a value of 255 (0FF Hex) in this register, all timers will be removed.

Since the two sub-functions "Install_Timer (4)" and "Install_Timer_Interrupt (6)" are functionally very similar, they can be saved by the TSR in the same buffer area. Thus, ID numbers returned by these two functions are unique: if "Install_Timer" assigns an ID of 2, then "Install_Timer_Interrupt" cannot use that ID number until it is released by "Remove_Timer." In addition, even though these two functions work out of the same buffer, their corresponding remove functions (5 and 7) can only remove timers that have been assigned by their own install functions.

4.9 Restart Timer—This function restarts a 16-bit timer to its preassigned (initial) value.

Call with:

AH = 0 = Function: Timer
AL = 8 = Sub-Function: Restart Timer
CL = Timer ID#

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function out of range
5 = ID does not exist
7 = Not initialized

Coding Examples:

Assembly Language:

```
Timer_ID   DB 0                          ;
                                         ;

MOV  AX,0008H                            ; Function/Sub-function = Restart Timer
MOV  CL,Timer_ID                         ; ID number
INT    62H                               ; Restart timer
;      Error checking here               ;
                                         ‘
```

C Language:

```
/*
 *Include the C libraries for REGS definition.   Define interrupt level.
/*
     #include    <dos.h>
     #define TSR_Intr 0x62
     union REGS inregs, outregs;
     unsigned char Timer_ID;
     {
     inregs.h. cl  = Timer_ID;           /*   Load the timer's identification number   */
     inregs.x.ax = 0x0008;               /*   Function/Sub-function = Timer Restart    */
     int86x(TSR_Intr, &inregs, &outregs);/*   Gateway to TSR interface                 */
     /*    Error checking here           */                                            */
     }
```

NOTE—**Timer_ID#, CL**:—Identifies the timer or timer interrupt being referenced. This number, 8 bits unsigned, was assigned by the TSR at installation time. If a value of **0FF Hex** is sent in CL, then all timers and timer interrupts will have their values restarted.

5. Application Program Interface for SAE J1708 Communication Functions

5.1 Initialize SAE J1708 Channel—Initializes both the communications and timer interrupt sections of the TSR. Clears all memory, sets up vectors, and saves all information so as to allow a return to pre-initialization conditions.

Call with:

```
AH =   1   =   Function:  Communication
AL =   0   =   Sub-Function: Initialize SAE J1708 Channel
BX =       =   User Definable
```

Returns with:

```
AL =   0   =   Status OK
       1   =   Function value out of range
       2   =   Sub-function out of range
       8   =   Pre-assigned, or initialization done
```

Coding Examples:

Assembly Language:

```
                                    ;
    MOV   AX,0100H                  ; Function/Sub-function = Initialize Communications
    MOV  BX,0000H                   ; Be sure user definable register is cleared
    INT     62H                     ; Gateway to TSR functions
    ;     Error checking here       ;
                                    ‘
```

C Language:

```
*Include the C libraries for REGS definition.   Define interrupt level.
    #include <dos.h>
    union REGS inregs, outregs;
    {
    inregs.x.ax  = 0x0100;          /*   Function/Sub-function = Initialize SAE J1708*/
    inregs.x.bx = 0x0000;           /*   Be sure user definable register is cleared    */
    int86x(0x62, &inregs, &outregs);/*   Invoke interrupt interface                    */
    /*    Error checking here       */                                                }
```

NOTE—The hardware used for the communications needs to remain completely transparent to the application programmer. Thus, any initialization or set up unique to the specific hardware being utilized must be done during installation of the TSR itself, in the resident portion of the program. Items such as port selection and addressing, memory allocation, and initialization for each configuration (vector assignments, etc.) needs to remain as part of the TSR transient program.

The timer interrupt it set up and initialized as part of this function call. There is no need to make a separate call to the timer initialization (Function = 0, Sub-Function = 0).

5.2 Reset SAE J1708 Channel—Clears error conditions, re-initializes communications port, and clears receive and transmit buffers. Does not purge existing timers or installed communications filters.

Call with:

```
AH =   1   =   Function:  SAE J1708 Communication Module
AL =   1   =   Sub-Function: Reset Channel
```

Returns with:

```
AL =   0   =   Status OK
       1   =   Function value out of range
       2   =   Sub-function out of range
       7   =   No-exist, communication has not been initialized
```

Coding Examples:

Assembly Language:

```
                                    ;
    MOV  AX,0101H                   ; Function/Sub-function = Reset Communications
    MOV  BX,0000H                   ; Be sure user definable register is cleared
    INT    62H                      ; Gateway to TSR interface
    ;     Error checking here       ;
```

C Language:

```
/*
 *Include the C libraries for REGS definitions.  Define interrupt level.
*/
    #include <dos.h>
    union REGS inregs, outregs;
{
    inregs.x.ax  = 0x0101;          /*   Function/Sub-function = Reset SAE J1708   */
    inregs.x.bx = 0x0000;           /*   Be sure user definable register is cleared  */
    int86x(0x62, &inregs, &outregs); /*   Invoke interrupt interface                */
    /*   Error checking here                                                        }
```

NOTE—This function does a soft reset of the communications port. It does not purge any timers, buffers, filters, etc., that have not been previously installed. It is used as an error condition recovery. If a clearing of the timers is desired, use Function = 0, Sub-function = 1.

5.3 Restore DOS Handler—This function restores the timer and communications to their pre-initialization condition. It does not eliminate the TSR Program; however, the TSR is dormant and will require re-initialization before it is usable again.

Call with:

AH = 1 = Function: Initialize SAE J1708 Channel
AL = 2 = Sub-Function: Reset DOS Handler
BX = User Definable (must be set to zero if not used for specific user function)

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function out of range
7 = No-exist, communication has not been installed

Coding Examples:

Assembly Language:

```
                                ;
    MOV  AX,0102H               ; Communication, Restore DOS
    MOV  BX,0000H               ; Be sure user definable register is cleared
    INT   62H                   ; TSR interface
    ;    Error checking here    ;
```

C Language:

```
    #include <dos.h>
    union REGS inregs, outregs;
{
    inregs.x.ax  = 0x0102;          /*   Function/Sub-function = Restore DOS    */
    inregs.x.bx = 0x0000;           /*   Be sure user definable register is cleared*/
    int86x(0x62, &inregs, &outregs); /*   Invoke interrupt interface             */
    /*   Error checking here        */                                          }
```

5.4 Retrieve SAE J1708 Information—To determine the usability of a given driver, it is necessary to have access to certain information relating to the TSR itself. This module returns information relating to the manufacturer, hardware used, SAE time standards compatibility, version number, etc.

Call with:

```
AH =  1  =  Function: Communications
AL =  3  =  Sub-Function: TSR program information
CX =        Size in bytes of buffer area pointed to by ES:DI
ES:DI==>    Segment and offset where information is to be stored
```

Returns with:

```
AL =  0  =  Status OK
      1  =  Function value out of range
      2  =  Sub-function out of range
      3  =  Compatibility information not supported by TSR
      4  =  Insufficient room in buffer area
```

Coding Examples:

Assembly Language:

```
                                    ;
    Com_Info DB 100 Dup(?)          ; Buffer area for TSR information storage
    Info_Size EQU $ – Com_Info      ; Size, in bytes, of buffer area
    MOV  AX,Seg Com_Info            ; Load segment where buffer resides
    MOV  ES,AX                      ; and move it into the extra-segment.
    MOV  DI,OffSet Com_Info         ; Load offset of buffer address.
    MOV CX,Info_Size                ; Load size, in bytes, of buffer into CX.
    MOV AX,0103H                    ; Function/Sub-function =Get TSR Information
    INT   62H                       ; Gateway to TSR interface handlers
                                    ;
    ;    Error checking here        ;
```

C Language:

```
/*
 *include C libraries for REGS definitions.   Define interrupt level.
 * and register memory locations.
*/
    #include <dos.h>
    #define  TSR_Intr 0x62
    union REGS inregs, outregs;
    struct SREGS segregs:
/*
 * Define a structure with the elements that are to be returned in the information block. The*
 * items in the structure below are compatible with Version 1.00 and above of this recommended.*
 * practice.  Support for versions beyond 1.00 may use a union to hardcode in definitions to*
 * accommodate a variety of later additions.  This structure, however, will remain constant as is*
 * the array element called “Reserved.”                                                    *
/*
    struct TSR_Info:
    {
    unsigned int      Buffer_Size          /*  Valid data byte count          */
                      Manufacture_Version, /*  Manufacture specific           */
                      SAE_Version;         /*  SAE standards support          */
```

```
    char            Author_ID[5],       /*  TSR software developer          */
                    Hardware[3];        /*  Hardware supported by TSR       */
    unsigned int    DOS_Version,        /*  MS-DOS Version support          */
                    Rx_Filter_Capacity, /*  Receive filters allowed         */
                    Rx_Filter_Level,    /*  Receive filters available       */
                    Tx_Filter_Capacity, /*  Transmit filters allowed        */
                    Tx_Filter_Level,    /*  Transmit filters available      */
                    Reserved[20];       /*  For future expansion            */
                    User_Define[32];    /*  User definable                  */
}
struct TSR_Info Com_Info;
struct TSR_Info far *Com_Ptr;
{
Com_Ptr = &Com_Info;                    /*  Load structure address into a far pointer */
segregs.es = FP_SEG(Com_Ptr);           /*  and use it to find the segment  */
inregs.x.di = FP_OFF(Com_Ptr);          /*  and the offset of structures address.  */
inregs.x.cx = sizeof(Com_Infor);        /*  Load size of structure in bytes  */
inregs.x.ax = 0x0103;                   /*  Function/Sub-function = TSR Information*/
int86x(TSR_Intr, &inregs, &outregs, &segregs);/*                             */
                                        /*  TSR interface gateway           */
/*  Error checking here                 */                          }
```

NOTE—

Size, CX:—Register CX contains a binary count of the size of the valid buffer area. Its purpose is to prevent later versions of the standard from overwriting the application code's memory beyond the memory allocated to this buffer.

Buffer_Seg, ES:—The segment in the application program's memory where the information parameters are to be put by the TSR.

Buffer_Off, DI:—This is the offset of this buffer area. The buffer contents that ES:DI point to are as shown in Table 5.

TABLE 5—INFORMATION BLOCK—COMMUNICATION SECTION

Offset	Label	Description
00H-01H	Buffer Size	Byte count of valid data in buffer area. Does not count itself.
02H-03H	TSR Version	TSR program version number, manufacture specific. a.b.: a = byte 03H, b = byte 02H, binary.
04H-05H	SAE Standard	Version of SAE standard fully supported, a.b.: a = byte -5H, b =Byte 04H, binary. Initial version 1.00.
06H-0AH	Author	Manufacture author of program (5 Character ASCII).
0BH-0DH	Hardware	Hardware that the software is compatible with (3 character ASCII).
0EH-0FH	DOS Version	Minimum DOS version supported by the TSR program. a.b.: a = byte 0FH, b = byte 0EH, binary.
10H-11H	Rx Capacity	Total number of receive filters that can be installed at any one time.
12H-13H	Rx Level	Number of filters currently installed.
14H-15H	Tx Capacity	Total number of transmit buffers that can be installed at any one time.
16H-17H	Tx Level	Number of transmit buffers currently installed.
18H-3FH	Reserved	Reserved for future assignment by SAE.
40H-7FH		User definable.

5.5 Install Usage Buffer—The amount of activity on the bus is continuously monitored by the TSR. It tracks the percentage of bus utilization and the percentage of bus errors, both current and average. These 4 parameters are each in an unsigned word, 16-bits. The value returned is the percentage divided by 100. The formulas for calculating these percentages are shown in Equations 1 and 2:

$$\% \text{ Utilization} = \frac{(\text{Total Characters} + 2 * \text{Total Packets}) * 10.417}{\text{Time Period}} \bullet 100 \qquad \text{(Eq. 1)}$$

$$\% \text{ Bus Error} = \frac{(\text{Error Characters} + 2 * \text{Error Packets}) * 10.417}{\text{Time Period}} \bullet 100 \qquad \text{(Eq. 2)}$$

Errors are bit collisions, incorrect CHECKSUMs, incorrect packet size, etc.

The update frequency is set at 0.651 s with an averaging period of 10.417 s. The percentages stored within the buffer area are updated automatically per update frequency period. Additionally, the first word in the buffer area is the bus status flag. The flag gives current activity status as to bus performance along with an update indicator.

The segment and offset of the two percentages and the flag are to be sent into this task. It will then update these parameters automatically as to the command periods.

Call with:

AH	=	1	= Function: Communications
AL	=	4	= Sub-Function: Set up utilization buffer
ES	=		Segment register of flag and buffer
DI	=		Address of buffer area where results are stored

Returns with:

AL	=	0	= Status OK
		1	= Function value out of range
		2	= Sub-function out of range
		3	= Request is not supported
		7	= SAE J1708 not initialized yet
		8	= Already installed

Coding Examples:

Assembly Language:

```
                                        ;
                                        ; Install a 16-byte buffer to hold utilization results in.
                                        ; Set the control flag to an update period of 0.651.
                                        ; seconds with an averaging period of
                                        ; 10.417 s.
Util_Info    DB    16 Dup(?)            ; Buffer area for TSR information storage
MOV  AX,Seg Util_Buffer                 ; Load segment where buffer resides
MOV  ES,AX                              ; and move it into the extra-segment.
MOV   DI,OffSet Util_Buffer             ; Load offset of buffer address.
MOV   AX,0104H                          ; Function/Sub-function = Bus Utilization
INT     62H                             ; Gateway to TSR interface handlers
;     Error checking here               ;
```

C Language:

```
/*
*    Include the C libraries for REGS definitions.  Define interrupt level and
*    Register memory locations
*/

     #include   <dos.h>
     #define    TSR_Intr 0x62
     union  REGS  inregs, ouregs;
     struct SREGS  segregs;

/*
*    Define a structure with the elements that are to be loaded by the TSR into our buffer.
*/

     struct Util_Info
     {
     unsigned int      Bus_status,              /*   Bus status bits
                       Util_Average,            /*   Utilization average
                       Error_Average,           /*   Bus error average
                       Util_Instantaneous,      /*   Current utilization rate
                       Error_Instantaneous,     /*   Current bus error rate
                       Reserved[3];             /*   For future expansion
     };
     struct Util_Info Util_Buffer;
     struct Util_Info far*Util_Ptr;
     {
     Util_Ptr = &Util_Buffer:                   /*   Load structure address into a far        */
     segregs.es = FP+SEG(Util+Ptr);             /*   pointer & use it to find the segment     */
     inreg.x.di = FP_Off(Util_Ptr);             /*   and the offset of structures address.    */
     inregs.x.ax = 0x0104;                      /*   Function/Sub-function = Utilization      */
     int86x(TSR_Intr, &inregs, &outregs, &segregs);
                                                /*   Gateway to TSR interface                 */
/*   Error checking here                        */
```

NOTE—As shown in Table 6, Bit0 through Bit7 are failure indicators. One of these bits being set means that the TSR is unable to gain access to the bus at all. Bit8 through Bit14 are indicators of a higher-than-desired error rate. It means the TSR is able to access the bus but the error rate is high enough to merit investigation. The error percentage for these bits is above 5%. Bit15 is set to 1 each time the percentages are update. The new averages are calculated every update period.

TABLE 6—BUS USAGE BUFFER

Offset	Label	Description
00H-01H	Bus Status	Bus operational status: Failures: Bit0: access, bus too busy Bit1: access, bus appears dead Bit2: no echo-back characters Bit3: controller hardware failure Bit4: user definable Bit5: user definable Bit 6: user definable Bit 7: unable to recover Warnings: Bit8: access, bus utilization rate Bit9: high collision rate Bit10: high echo-back failure rate Bit11: reserved for future SAE assignment Bit12: user definable Bit13: user definable Bit14: user definable Bit15: updated
02H-03H	Utilization Average	Bus utilization average of most recent update period
04H-05H	Error Average	Bus error percentage average for most recent update period
06H-07H	Utilization	Bus utilization average over more recent averaging period
08H-09H	Error	Bus error percentage average for most recent averaging period
0AH-0FH	Reserved	Reserved for future SAE assignment

5.6 Remove Usage Buffer—This function removes the usage buffer allocated by Sub-function 4. Releases the buffer back to the application program's memory pool.

Call with:

AH = 1 = Function: Communications
AL = 5 = Sub-Function: Remove Usage Buffer

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function out of range
7 = No buffer has been allocated

Coding Examples:

Assembly Language:

```
;
MOV  AX,0105H          ; Function/Sub-function = Remove Usage Buffer
INT   TSR_Intr         ; Gateway to communications interface
;    Error checking here ;
```

C Language:

```
/*
 *Define a structure with the elements that are to be loaded by the TSR into our buffer.
/*
    #include <dos.h>
    union REGS inregs, outregs;
    {
    inregs.x.ax  = 0x0105;                  /*   Function/Sub-function = Remove usage buffer*/
    int86x(TSR_Intr, &inregs, &outregs, & segregs);                                        */
                                            /*   Gateway to TSR interface                  */
    /* Error checking here                  /*                                             }
```

5.7 Install MID/PID Receive Filter—This function sets up a receive filter located in the application program's memory area. Dual buffering is used to handle arbitration between the TSR and application program. A control flag dictates the filtering and buffer assignment. The buffers and control/status flag are passed in registers as addresses along with their memory segment.

Once the end of the message is received, the packet is checked to be sure its CHECKSUM is correct and its size is within limits. If the packet is correct, then the data is broken down into its individual PID members. It is then compared to the installed receive buffers for a match. It must match on all of the specifications designated in the command portion of the flag. If the criteria for all of the command bits are met, and there is enough room in the buffer, then it is loaded into the data buffer in the format shown in Table 7. If no match is found, then the message is discarded.

It should be noted that multiple filters with the same or similar configurations can be installed at the same time.

As messages are received, they are loaded into the filter buffers that match them. Bit8 of the control/status flag indicates which buffer the TSR is to put the data into. Once data has been loaded into the filter buffer, the TSR will then set Bit9 (Data Loaded) of the flag. The application program needs to monitor Bit9 in order to know when data is available. At which point, the application program can set Bit7 (switch buffers) of the flag and wait for the TSR to switch buffer assignments. A worst case switch will occur within 150 ms. When the TSR performs the buffer switch, it clears Bit7 (switch buffer command) and inverts the buffer assignments, Bit8. In the event of a buffer overflow, the TSR resets its pointers, overwrites older data, and sets Bit10 (overflow) of the flag.

If multiple message buffering is selected, then the TSR loads as many header/data combinations as room allows. They are loaded sequentially with the *byte count* following the last data byte of the previous message. When pulling data from such a buffer, a value of 0000H in the byte count indicates end of valid data.

TABLE 7—RECEIVE BUFFER

Offset	Label	Description
00H-01H	Byte Count	Byte count is an unsigned integer (16 bits). The number of bytes in this packet, not counting itself.
02H	Message Status	Status: Bit0 = 1 = Echo-back Bit1 – Bit 7= Reserved for future SAE assignment
03H		User definable
04H-07H	Timer Byte	When the first byte of any packet is received, the timer interrupt's count is saved. The resolution of the counter is in milliseconds and is 32 bits in length. When the packet is stored in the buffer, the timer of its arrival is also stored. This is the time that the first byte of the packet is received.
08H	MID	MID of incoming packet
09H	PID	First PID of packet received
0AH	Data	Data of MID and PID previously. Data up to 18 byte in length. If full packet is requested, this field may hold additional PID's data, and CHECKSUM is only saved for full packet request.

Call with:

AH	= 1	=	Function: Communications
AL	= 6	=	Sub-Function: Install receive filter
CX	=		Size in bytes of smaller buffer
DL	=		MID to be filtered by, if any
DH	=		PID to be filtered by, if any
ES	=		Segment where buffers and flag reside
BX	=		Address of control/status flag
DI	=		Address of Buffer #1
SI	=		Address of Buffer #2

Returns with:

AL	= 0	=	Status_OK
	1	=	Function value is out of range
	2	=	Sub-function value is out of range
	3	=	Request not supported by this driver.
	4	=	Insufficient room in TSR's buffer space (30 buffers maximum)
	9	=	Invalid parameter list.
AH	=		Buffer identification number

Coding Examples:

Assembly Language:

```
TSR_Intr    EQU 62H                 ; TSR interface interrupt level
Filter_Cmd  EQU 0010_1011B          ; Command value:  filter by MID & PID
                                    ; for echo packets;
                                    ; for echo packets, stack packets in buffer.
MID_Filter  EQU 128                 ; The MID we are filtering for.
PID_Filter  EQU 183                 ; The PID we are looking for.
Buf_Size    EQU 100                 ; Size in bytes of buffers.
                                    ;
Rx_ID       DB?                     ; Storage of TSR Assigned ID Number for this
                                    ; filter.
Rx_Flag     DW ?                    ; Control/Status flag of receive filter.
Buffer1     DB Buf_Size Dup(?)      ; Declare 2 receive buffers to switch back
Buffer2     DB Buf_Size Dup(?)      ; and forth between TSR.
                                    ;
MOV  Rx_Flag,Filter_Cmd             ; Load command flag with filter configuration
MOV  AX,Seg Buffer1                 ; Load the segment of the 2 buffers and flag
MOV  ES,AX                          ; and put it into the extra-segment register.
MOV  DI,Offset Buffer1              ; Load offset address of first buffer
MOV  SI,Offset Buffer2              ; and load the second buffer's address.
MOV  BX,Offset Rx_Flag              ; Load address of the control/status flag.
MOV  DH,PID_Filter                  ; Load the PID to filter by, if any,
MOV  DL,MID_Filter                  ; and the MID to filter by, if any.
MOV  CX,Buf_Size                    ; Load the size of the buffers.
MOV  AX,0106H                       ; Function/Sub-function = Set Rx filter
INT  TSR_Intr                       ; Gateway to communications interface
                                    ;
;    Error checking here            ;
                                    ;
MOV  Rx_ID,AH                       ; Save this filter's identification number.
```

C Language:

```
    #include     <dos.h>
    #define TSR_Intr 0x62                 /*  TSR interface interrupt level          */
    #define Filter_Cmd 0x002b             /*  Command value; Filter by MID and       */
                                          /*  PID for just echo-back packets,        */
                                          /*  stack packets in buffer.               */
    #define Buf_Size 100                  /*  Size in bytes of buffers               */
    #define MID_Filter 128                /*  MID we are filtering for.              */
    #define PID_Filter 183                /*  PID we are looking for.                */
    union REGS inregs, outregs;
    struct SREGS segregs;
    struct Rx_Filter*
    {
    unsigned int   Flag;
    unsigned int   Buffer1 [Buf_Size],
                   Buffer2 {Buf_Size],
    unsigned char ID;
    };
    struct Rx_FilterReceiver;
    struct Rx_Filterfar *Rx_Ptr;
    {
    Receiver.Flag = Filter_Cmd:           /*  Load command flag with filter.         */
    Rx_Ptr = &Receiver;                   /*  Put address in structure pointer       */
    segregs.es = FP_SEG(Rx_Ptr);          /*  Load the segment of structure in ES.   */
    inregs.x.bx = FP_OFF(Rx_Ptr);         /*  Load address of control/status flag    */
    inregs.x.di = inregs.x.bx + 2;        /*  Load offset address of first buffer    */
    inregs.x.si = inregs.x.di + Buf_Size; /*  and load the second buffers address.   */
    inregs.h.dh = PID_Filter;             /*  Load the PID to filter by, if any,     */
    inregs.h.dl = MID_Filter;             /*  and the MID to filter by, if any.      */
    inregs.x.cx = Buf_Size;               /*  Load the size of the buffers.          */
    inregs.x.ax = 0x0106;                 /*  Function/Sub-Function = Rx Filter      */
    int86x(TSR_Intr, &inregs,             /*                                         */
      &outregs, &segregs);                /*  Gateway to TSR interface               */
/*
* Error checking here                     /*
*/
    Receiver.ID = outregs.h.ah;           /*  Save this filter's identification number */
    }
```

NOTE—**Segment, ES**:—This is the segment register in the application program where the buffers and flag reside.

Flag, BX:—The flag is passed in as an address. The address or offset where the actual flag resides. The flag holds the instructions on how the buffer is actually to be setup. The lower byte is set up by the application programmer while the upper byte contains status information from the TSR as to the current condition. Following is a bit definition of this flag.

Bit0: MID Filter
If this bit is set equal to 1, then the received packets are filtered by type MID value. Otherwise, all MID's are loaded so long as they meet the rest of the requirements.

Bit 1: PID Filter
If this bit is set equal to 1, then the received packets are filtered by their PID value. Otherwise, all PID's are loaded so long as they meet the rest of the requirements.

Bit 2: Non-Echo Included
If this bit is set, then data received from outside this unit is loaded into the Receive buffer. As always, it must meet all other requirements. If this bit has a value of 0 or is cleared, no data coming from the outside is loaded regardless of other specifications.

Bit3: Echo Included
If this bit is set, then data that has been sent or transmitted by this TSR is loaded into the Receive buffer. As always, it must meet all other requirements. If it is cleared, then no echo-back data is loaded.

Bit4: Full Packets Included
When data meeting the listed requirements is found, the entire packet from MID to the CHECKSUM is loaded into the buffer. Otherwise, data being requested via the command bits are loaded.

Bit5: Multiple Message Buffering
If set, then as much data as will fit, is loaded into the buffers. If cleared, then only 1 item is loaded.

Bit6: User definable

Bit7: Switch Buffers
This bit is set by the application program when it needs more data. When operations are complete on the buffer that is currently held by the application program, it may set this bit to trigger the interrupt handler to switch buffer assignments. The interrupt handler only switches the buffers and clears this bit when there is data available.

Additionally, the interrupt handler periodically checks the switch bits no less than every 0.15 s to see if data is available and a switch is being requested. This is to assure the timely access to available data, even if no more data is being received.

Bit8: Interrupt Handler Has Buffer #1
This bit indicates to both the TSR and the application program as to which buffer each is allowed to access. A value of 1 means that the application program is free to access Buffer2. A value of 0 means it may access Buffer1.

Bit9: Data Loaded in Buffer
This bit being set indicates that there is data available in the buffer being held by the TSR. The application program may perform a switch to obtain rights to this data.

Bit10: Buffer Overflow
This bit being set means a buffer overflow has occurred on the buffer currently being held by the TSR. Data has been lost and overwritten. The most recent data are preserved.

Bit11: Reserved for future SAE assignment

Bit12: User definable

Bit13: User definable

Bit14: User definable

Bit15: User definable

Buffer1, Buffer2, and Size, DI, SI, CS—These are the buffers where the actual received data is to be stored. They require a minimum size of 11 bytes plus the data size. The maximum is limited to both buffers and flag fitting in one segment (65 536 bytes). The size parameter is a byte count of the size of the buffers. Each buffer is equal in size and must be of size “Size” or greater.

MID and PID, DX—These are the parameters that the TSR filters by. If Bit0 of the flag is set, then MID is used. If Bit1 of the flag is set, then PID is used. Either none, one, or both parameters may be filtered by in any one case.

5.8 Remove MID/PID Receive Filter—This function removes the receive filter from the TSR. It releases the TSR's memory assignment for the receive buffers. It releases the buffers back to the application program. The buffer ID that was returned by the TSR in Sub-Function 6, "Install Receive Filter," must be specified.

Call with:

AH = 1 = Function: Communication
AL = 7 = Sub-Function: Remove MID/PID Receive Filter
CL = ID number of filter to be disabled

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function out of range
5 = Invalid ID, or ID not found in current list
7 = No-exist, TSR has not been initialized

Coding Examples:

Assembly Language:

```
;
TSR_Intr  EQU 62H                ; TSR interface interrupt level
Buf_Size  EQU 100                ; Size, in bytes, of buffers
                                 ;
Rx_ID     DB ?                   ; Storage of TSR assigned ID number for this filter
Rx_Flag   DW ?                   ; Control/Status flag of receive filter
Buffer1   DB Buf_Size Dup(?)     ; Declare 2 receive buffers to switch back
Buffer2   DB Buf_Size (Dup?)     ; and forth between TSR.
MOV    CL,Rx_ID                  ; Load the filter's identification number.
MOV     AX,0107H                 ; Function/Sub-function = Remove Rx Filter
INT    TSR_Intr                  ; Gateway to communications interface
                                 ;
;    Error checking here         ;
```

C Language:

```
#include      <dos.h>
#define TSR_Intr  0x62                /*  TSR interface interrupt level          */
#define Buf_Size  100                 /*  Size in bytes of buffers               */
union REGS inregs, outregs;
struct SREGS segregs;
struct Rx_Filter*
{
unsigned int      Flag;*
unsigned char     Buffer1 [Buf_Size],
                  Buffer2[Buf_Size];
unsigned char     ID;
};
struct Rx_Filter Receiver;
{
outregs.h.cl = Receiver.ID;           /*  Load the filter's identification number.  */
inregs.x.ax = 0x0107;                 /*  Function/Sub-function = Remove Rx filter  */
int86x(TSR_Intr, &inregs, &outregs);  /*  Gateway to TSR interface                  */
/* Error checking here                */
}
```

5.9 Install Transmit Broadcast Buffer—This function sets up a transmit buffer in the application program's memory area. The TSR loads this data from the buffer into its internal memory when it's time for the message to be sent. It then builds a packet along with other messages of the same MID. The TSR calculates the CHECKSUM and then transmits the packet. It operates on a base frequency of 100 ms. Byte count, shown in Table 8 represents the size of the MID, PID, and data field. It does not include the CHECKSUM or itself. No checking or verification of the PID to the data is done.

TABLE 8—TRANSMIT BROADCAST BUFFER

Offset	Label	Description
00H-01H	Byte Count	Byte count is an unsigned integer (16 bits). The number of bytes in a packet, not counting itself.
02H	MID	MID of outgoing packet.
03H	PID	First PID of the packet.
04H-15H	Data	Data of MID and PID previously. Data up to 18 bytes in length. This field may hold additional PIDs and data.

Call with:

AH = 1 = Function: Communications
AL = 8 = Sub-Function: Install Transmit Buffer
CL = Frequency of transmissions
CH = Message priority
ES = Segment where buffers and flag reside
BX = Address of control/status flag
DI = Address of buffer

Returns with:

AL	=	0	= Status OK
		1	= Function value out of range
		2	= Sub-function out of range
		4	= Insufficient room in TSR's buffer space
		7	= No buffer has been allocated
		9	= Invalid parameter list
AH	=		Buffer's identification number

Coding Examples:

Assembly Language:

```
                                        ;
TSR_Intr        EQU 62h                 ; TSR interface interrupt level
Filter_Cmd      EQU 0000_0101B          ; Command value:  Active and repeat transmissions
Frequency       EQU 1                   ; Repeat message every 100 ms.
Buf_Size        EQU 22                  ; Size in bytes of buffers
                                        ;
Tx_ID           DB ?                    ; Storage of TSR assigned ID number
Tx_Flag         DW ?                    ; Control/Status flag of transmissions
Buffer          DB Buf_Size Dup(?)      ; Transmitter buffer
                                        ;
MOV             Tx_Flag,Filter_Cmd      ; Load command flag with transmitter configuration.
MOV             AX,Seg,Buffer           ; Load the segment of the buffer and flag
MOV             ES,AX                   ; and put it into the extra-segment register.
MOV             DI,OffSet Buffer        ; Load offset address of transmitter buffer.
MOV             BX,Offset Tx_Flag       ; Load address of the control/status flag.
MOV             CX,Frequency            ; Load the transmission repeat frequency.
MOV             AX,0108H                ; Function/Sub-function = Install Tx filter
INT             TSR_Intr                ; Gateway to communications interface
                                        ;
;    Error checking here                ;
                                        ;
MOV             Tx_ID,AH                ; Save this transmitter's identification number.
                                        ;
```

C Language:

```
#include <dos.h>
#define TSR_Intr0x62                    /*  TSR interface interrupt level          */
#define Filter_Cmd 0x0005               /*  Command value:  repeat transmission    */
#define Buf_Size 22                     /*  Size in bytes of buffers               */
#define Frequency 1                     /*  Frequency of transmission repeats      */
#define Priority 8                      /*  Access priority for message            */
Union REGS inregs, outregs;
struct SREGS segregs;
struct Tx_Group
{
unsigned int        Flag;
unsigned char       Buffer[Buf_Size];
unsigned char       ID;
};
struct Tx_Group     Transmitter;
struct Tx_Group     far *Tx_Ptr;
{
```

```
Transmitter.Flag = Filter_Cmd;      /*  Load command flag.                  */
Tx_Ptr = &Transmitter;              /*  Put address in structure pointer    */
segregs.es = FP_SEG(Tx_Ptr);        /*  Load the segment of structure in ES.*/
inregs.x.bx = FP_OFF (Tx_Ptr);      /*  Load address of control/status flag.*/
inregs.x.di = inregs.x.bx + 2;      /*  Load offset address of buffer.      */
inregs.h.cl = Frequency;            /*  Load the frequency of the buffers.  */
inregs.h.ch = Priority;             /*  Load message bus access priority.   */
inregs.x.ax = 0x0108;               /*  Function/Sub-function = Tx filter.  */
int86x(TSR_Intr, &inregs,           /*                                      */
  &outregs, &segregs);              /*  Gateway to TSR interface            */
/* Error checking here              /*
Transmitter.ID = outregs.h.ah;      /*  Save this transmitter's ID number   */
}
```

NOTE—

Segments, ES:—This is the segment register in the application program where the buffers and flag reside.

Flag, BX—The flag is passed in as an address: the address or offset where the actual flag resides. The flag is the key instruction on how the buffer is actually to be setup. Following is a bit definition of this flag.

Bit0: Transmit Message is Active
Indicates that the message is being or soon will be sent or transmitted. This bit can be set or cleared by the application program to start or stop transmitting. If message is a oneshot, Bit2 = 0, then this bit is cleared by the TSR each time the data has been read into transmit buffer.

Bit 1: Remove Transmit Buffer from List
Command to TSR to remove buffer from its active list and release the memory. If this is set on installation, then the buffer is automatically released after first transmission.

Bit 2: Repeat Message
If set, then the message is repeated or retransmitted at the rate specified in frequency. If cleared, then the message is sent just once with the value in frequency used as a delay before transmitting.

Bit3: No Packet Build
If this bit is set, then this message is not combined with any other messages. It is sent out in a packet by itself. If not set, then the TSR will attempt to combine message into as large a packet (21 bytes or less) as possible.

Bit4: Reserved for future SAE assignment

Bit5: User definable

Bit6: User definable

Bit7: Busy—Application Program in Buffer
This bit should be set by the application program anytime it is writing to the buffer. When the TSR goes to load the data from the buffer, it first checks this bit to avoid any conflict.

Bit8: Transmitted—Message Was Sent
This bit is set each time the data is transferred into the TSR memory.

Bit9: Removed Buffer From List
When a request to remove a transmit buffer is received and acted upon by the TSR, it sets this bit upon completion.

Bit10: Reserved for future SAE assignment

Bit11: Reserved for future SAE assignment

Bit12: User definable

Bit13: User definable

Bit14: User definable

Bit15: User definable

Buffer—This is the buffer in the application program's memory where the message is stored. It follows the format, previously. The data can be altered or changed by the application program, as desired. Whenever the application program is writing to this area for more than one instruction, it needs to set the "Busy" bit of the flag. This is to avoid the TSR pulling the data out of the buffer when it has been partially altered.

Frequency, CL—This is the frequency at which the message is to be transmitted. It represents a resolution of 1 count for each 100 ms. When this routine is called and the transmit buffer is loaded, the TSR uses this as the countdown to its first transmission attempt. After that, if the message is a repeat message, then this value represents the time between transmissions. It is a decrement timer. When the timer reads zero, the message is broadcast.

Priority, CH (Optional)—This is the bus access priority level. It is individually selective for each message to be transmitted. Values range from the highest priority (1) to the lowest (8). The default value is 8. This parameter is optional due to the different hardware configurations.

Buffer ID—The buffer ID is a way for the application program to access and identify this particular filter buffer at a later point. If successful at installing this buffer, the TSR returns, along with the status, an identification number. This number is a binary value that is used to identify this buffer for commands, such as removal of this buffer.

5.10 Remove Transmit Broadcast Buffer—This function releases the memory assignment of the transmit buffer. It releases the buffers back to the application program. The buffer ID must be specified.

Call with:

AH = 1 = Function: Communication
AL = 9 = Sub-Function: Remove Transmit Buffer
CL = ID number of buffer to be disabled

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function out of range
7 = Invalid ID, or ID not found in current list

Coding Examples:

Assembly Language:

```
                                    ;
TSR_Intr    EQU 62h                 ; TSR interface interrupt level
Tx_ID       DB ?                    ; Storage of TSR assigned ID number for this filter
                                    ;
MOV         CL,Tx_ID                ; Load the transmitter's identification number.
MOV         AX,0109H                ; Function/Sub-function = Remove broadcast buffer
INT         TSR_Intr                ; Gateway to communications interface
;   Error checking here             ;
```

C Language:

```
#include   <dos.h>
#define TSR_Intr0x62                /*  TSR interface interrupt level              */
union REGS inregs, outregs;
struct Tx_Group
{
unsigned int        Flag;
unsigned char       Buffer[Buf_Size];
unsigned char       ID;
};
struct Tx_Group   Transmitter;
{
outregs.h.cl = Transmitter.ID;      /*  Load the filter's identification number    */
inregs.x.ax = 0x0109;               /*  Function/Sub-function = Remove broadcast.  */
                                    /*  buffer                                     */
int86(TSR_Intr, &inregs &outregs,); /*  Gateway to TSR interface                   */
/* Error checking here              */                                             }
```

5.11 Install Transmit Message—This function loads the contents of a buffer into the TSR's memory area for output at the next 100 ms period. It is a simplified subset of Sub-Function 8, "Install Transmit Broadcast Buffer." Frequency, Flag, and Buffer_ID are not used. No removal counterpart to this command is required. The message is sent out once and then purged from the TSR memory automatically. Byte count, shown in Table 9, represents the size of the MID, PID, and data field. It does not include the CHECKSUM or itself. No checking or verification of the data is done.

TABLE 9—TRANSMIT MESSAGE BUFFER

Offset	Label	Description
01H	Byte Count	Byte count is an unsigned integer (16 bits). The number of bytes in a packet, not counting itself.
02H	MID	MID of outgoing packet.
03H	PID	First PID of the packet.
04H-15H	Data	Data of MID and PID previously. Data up to 18 bytes in length. This field may hold additional PIDs and data.

Call with:

```
AH =  1   =  Function: Communications
AL =  10  =  Sub-Function: Install Transmit Buffer
CH =         Message priority (Optional)
ES =         Segment where buffers and flag reside
DI =         Address of buffer
```

Returns with:

AL = 0 = Status OK
1 = Function value out of range
2 = Sub-function out of range
4 = Insufficient room in TSR's buffer space
7 = No buffer has been allocated.
9 = Invalid parameter list

Coding Examples:

Assembly Language:

```
;
TSR_Intr    EQU 62h          ; TSR interface interrupt level
Priority    EQU 8            ; Bus access priority for message
                             ;
BufferDB 22 Dup(?)           ; Transmitter buffer
                             ;
MOV   AX,Seg Buffer          ; Load the segment of the buffer
MOV   ES,AX                  ; and put it into the extra-segment register.
MOV   DI,OffSet Buffer       ; Load Offset address of transmitter buffer.
MOV   CH,Priority            ; Load the transmission priority of message.
MOV   AX,010AH               ; Function/Sub-function = Install Transmit Message
INT   TSR_Intr               ; Gateway to communications interface
;   Error checking here      ;
```

C Language:

```
/*
 *  Define a structure with the elements that are to be loaded by the TSR into our buffer.
 */
    #include <dos.h>
    #define TSR_Intr0x62              /*  TSR interface interrupt level                */
    #define Priority 8                /*  Frequency of transmission repeats*/
    union REGS inregs, outregs;
    struct SREGS segregs;
    unsigned char Buffer[22];
    {
    segregs.es = FP_SEG(Buffer);      /*  Load the segment of the buffer into ES.      */
    inregs.x.di = FP_OFF(Buffer);     /*  Load offset address of buffer.               */
    inregs.h.ch = Priority;           /*  Load the message bus access priority.        */
    inregs.x.ax = 0x010A;             /*  Function/Sub-function = Transmit Message     */
    int86x(TSR_Intr, &inregs,         /*                                               */
      &outregs &segregs,);            /*  Gateway to TSR interface                     */
    /*      Error checking here       /*                                               }
```

NOTE—

Segment, ES—This is the segment register in the application program where the buffer reside.

Buffer, ES:DI—This is the buffer in the application program's memory where the message is stored. It follows the format previously. The data can be altered or changed by the application program as desired. The message is sent only once for each call to this command. Thus, if it is desired to retransmit this message, then multiple calls to this function are required.

Priority, CH—(Optional) This is the bus access priority level. It is individually selective for each message to be transmitted. Values range from the highest priority (1) to the lowest (8). The default value is 8. This parameter is optional due to the different hardware configurations.

5.12 Transmit Buffer Immediately—This function transmits the contents of a buffer over the SAE J1708 network. The segment and offset of a buffer, residing in the application program's memory area, is loaded into the TSR memory along with the offset of a control/status flag and a character counter. It should be noted that, as with the other functions, all three offsets must reside in a single segment. The contents of the buffer are transmitted over the network as soon as the bus access can be gained. There is no delay or 100 ms window of operation. The buffer contains data only, with no size word or header of any kind. The TSR will not alter this buffer in any way. No check sum is calculated. Also, there is no MID or PID breakdown or packet building: the buffer is sent as is. The data size can be up to 65 532 bytes, 4 bytes short of a full segment. These 4 bytes are required for the flag and counter storage (2 bytes each).

Call with:

AH	=	1	= Function: Communications
AL	=	11	= Sub-Function: Transmit buffer immediately
CH	=		Message priority (optional)
DX	=		Size of data field in buffer
ES	=		Segment where buffers, flag, and counter reside
BX	=		Address of control/status flag
DI	=		Address of buffer
SI	=		Address of byte transmit counter

Returns with:

AL	=	Return status
0	=	Status_OK
1	=	Function value out of range
2	=	Sub-Function value out of range
7	=	No buffer has been previously allocated
9	=	Invalid parameter list

NOTE—

Segment–ES—Segment register in the application program where the Buffer, Flag, and Counter reside.

Flag–BX—The flag is passed in as an address: the address, or offset, where the actual flag resides. The flag is the key or set of instructions on how the buffer is actually set up. Following is a bit definition of this flag.

Bit0: Reserved for future SAE assignment

Bit1: Reserved for future SAE assignment

Bit2: Reserved for future SAE assignment

Bit3: Reserved for future SAE assignment

Bit4: User definable

Bit5: User definable

Bit6: User definable

Bit7: User definable

Bit8: Transmitting
This bit is set during the transmission of the buffer. It stays set until the buffer transmission is completed

Bit9: Transmitted, Done
This bit is set when the transmission is completed successfully and the buffer has been released back to the application program.

Bit10: Error Recovery
Set when an error in the transmission is encountered. This bit is not cleared by the TSR once set until the buffer is transmitting successfully.

Bit11: Reserved for future SAE assignment

Bit12: User definable

Bit13: User definable

Bit14: User definable

Bit15: User definable

Buffer–DI—This is the offset of the buffer residing in the application program's memory area. Data is stored in binary format. The buffer contains data only, with no header or trailer. The TSR does no manipulation of the data; it is the responsibility of the application programmer to prepare the buffer completely.

Message Priority–CH—(Optional) This is the bus-access priority level. It can be set individually for each buffer being transmitted. Values range from; 1 (= highest priority) to 8 (= lowest priority). The default value is 8. Each level in the priority value represents 2 bit times after the idle time.

Transfer Count–SI—This is an address (offset) located in the application program's memory area. More specifically, it is located in the same segment as the flag and buffer. As the data is transmitted over the network, the word that is pointed to by this address is incremented from zero up. The application program may monitor this location to see how the buffer transmission is proceeding.

Data_Size–DX—This is the byte count of the data contained in the buffer. Its value can range from 1 to 65 532. The TSR will transmit non-stop the number of bytes in DX from the buffer. A value greater than 65 532 or a value of 0 is not allowed.

5.13 Install Receive Error Buffer—This function sets up a receive filter in the application program's memory area. Dual buffering is used to handle arbitration between TSR and application program. A control flag dictates the filtering and buffer assignment. The buffers and control status flag are passed in registers as addresses, both segment and offset. Size of the buffer is passed in register CX.

When the end of a message is received, the packet is checked to be sure its CHECKSUM is correct and its size is within established size limits. If the packet fails to meet these or any other imposed requirements and is found to be faulty, it is loaded into one of the buffers passed in via this function. This function is very similar to Function 1, Sub-Function 6, "Install Receive Filter Buffer." The purpose is to allow access to faulty data.

It should be noted that multiple filters with the same or similar configurations can be installed at any given time. The structure of the data within the buffers is shown in Table 10.

TABLE 10—RECEIVE ERROR BUFFER

Offset	Label	Description
00H-01H	Byte Count	Byte count is an unsigned integer (16 bits). Number of Bytes in this packet not counting itself.
02H	Message Status	Bit0 = Echo Back Bit1-Bit3 = Reserved for future SAE assignment Bit4-Bit7 = User definable
03H	Reserved	Reserved for future SAE assignment
04H-07H	Timer Value	When the first byte of packet is received, the timer interrupt count is saved. A 4-byte value with the least significant byte is first. 1 ms resolution.
08H-xx	Data	The entire message is stored here. Length is determined by internal receive buffer sizes.

Call with:

AH = 1 = Function: Communications
AL = 12 = Sub-Function: Install Receive Error Buffer
CX = Size in bytes of each buffer
ES = Segment where buffers and flag reside
BX = Address of control/status flag
DI = Address of Buffer #1
SI = Address of Buffer #2

Returns with:

AL = 0 = Status_OK
1 = Function value out of range
2 = Sub-function out of range
3 = Request is not supported by this driver
4 = Insufficient room in TSR's buffer space
7 = No buffer has been previously allocated
9 = Invalid parameter list
AH = Buffer's identification number.

Coding Examples:

Assembly Language:

```
;
sErr_Buf        EQU 200                 ; Size in bytes of buffers
Err_Command     EQU 0010_1100B          ; Non-echo, echo, and multiple buffering
Err_Flag        DW ?                    ; Control/status flag
Err_Buf1        DB sErr_BufDup(?)       ; Define 2 buffers for TSR to
Err_Buf2        DB sErr_BufDup(?)       ; switch between
Err_ID          DB ?                    ; Storage of TSR assigned ID, number of this
                                        ; buffer

MOV             Err_Flag,Err_Command    ; Set control flag
MOV             AX,Seg Err_Flag         ; segment address of buffers
MOV             ES,AX                   ; Flag into Extra Segment
MOV             BX,OffSet Err_Flag      ; Offset address of control flag
MOV             DI,OffSet Err_Buf1      ; Offset address of buffer #1
MOV             SI,OffSet Err_Buf2      ; Offset address of buffer #2
MOV             CX,sErr_Buf             ; Size of buffers in bytes
MOV             AX,010CH                ; Function/Sub-function = Install receive error
                                        ; buffer
```

```
        INT             62H                     ; Gateway to TSR interface
                                                ;
        ; Error checking                        ;
                                                ;
        MOV             Err_ID,AH               ; Save this buffers ID number
```

C Language:

```
        #include <dos.h>
        #define sErr_Buffer 200                 /*Size in bytes of buffers                          */
        #define Err_Command 0x2c                /*   Non-echo, echo & multiple buffering            */
        union REGS inregs, outregs;
        struct SREGS segregs;
        struct Err_Buffer
        {
        unsigned int        Flag                /*   Control/Status Flag                            */
        unsigned char Buffer1[sErr_Buffer],     /*   Define 2 buffers for                           */
        Buffer2[sErr_Buffer];                   /*   TSR to switch between                          */
        unsigned char ID;                       /*   Storage of TSR/assigned number of buffer       */
        };
        struct Err_Buffer Receiver;
        struct Err_Buffer far *Err_Ptr;
        {
        Receiver.Flag = Err_Command;            /*   Set control flag                               */
        Err_Ptr = &Receiver;                    /*                                                  */
        segregs.es = FPSET(Err_Ptr);            /*   Structures segment                             */
        inregs.x.bx = FP_OFF(Err_Ptr);          /*   Control Flags offset                           */
        inregs.x.di = inregs.bx + 2;            /*   Buffer1 address                                */
        inregs.x.si = inregs.di _sErr_Buffer;   /*   Buffer2 address                                */
        inregs.x.cx - sErr_Buffer;              /*   Size of error buffers                          */
        inregs.x.ax = 0x010c;                   /*   Function/Sub-function = Install Rx Error       */
        int86x(0x62, &inregs, &outregs, &segregs)/*  Gateway to TSR interface                       */
        Receiver.ID = outregs.h.ah;
        /* Error checking of return status      /*
        }
```

NOTE—

Segment–ES—This is the segment register in the application program where the buffers and flag reside.

Flag–BX—The flag is passed in as an address: The address, or offset, where the actual flag resides. The flag is the key or set of instructions on how the buffer is actually set up. The lower byte is set up by the application programmer, while the upper byte contains status from the TSR as to the current condition. Below is a bit definition of this flag.

Bit0: Reserved for future SAE assignment

Bit1: Reserved for future SAE assignment

Bit2: Non-Echo Included
If this bit is set then error data received from outside this unit will be loaded. As always, it must meet all other requirements. If this bit has a value of 0, cleared, no error data coming from the outside will be loaded regardless of other specifications.

Bit3: Echo Included
If this bit is set, then error data that has been sent or transmitted by this TSR will be loaded. If it is cleared, then no echo-back data will be loaded.

Bit4: Reserved for future SAE assignment

Bit5: Multiple Message Buffering
If set, then as much data as will fit will be loaded into the buffers. If cleared, then buffers will contain only 1 packet. In the event the buffer is not switched when full, new data will overwrite old, always starting packets at beginning of buffer.

Bit6: User definable

Bit7: Switch Buffers
This bit is set by the application program when it needs more data. When operations are complete on the buffer that is currently held by the application program it may set this bit to trigger the interrupt handler to switch buffer assignments. The interrupt handler will only switch the buffers and clear this bit when there is data available. Additionally, the interrupt handler will periodically check the switch bits, no less than 0.15 s, to see if data is available and a switch is being requested. This is to assure the timely access to available data even if no more data is being received.

Bit8: Interrupt Handler has Buffer #1
This bit indicates to both the TSR and the application program which buffer it is allowed to access. A value of 1 means that the application program is free to access buffer #2. A value of 0 means it may access Buffer #1.

Bit9: Data loaded in Buffer
This bit being set indicates that there is data available in the buffer being held by the TSR. The application program may perform a switch to obtain rights to this data.

Bit10: Buffer Overflow
This bit being set means a buffer overflow has occurred on the buffer currently being held by the TSR. Data has been lost and overwritten. Most recent data is preserved.

Bit11: Reserved for future SAE assignment

Bit12: User definable

Bit13: User definable

Bit14: User definable

Bit15: User definable

Buffer1, Buffer 2, & Size–DI, SI, CX—These are the buffer where the actual received data is to be stored. They require a minimum size of 9 bytes, plus the packet size. Maximum is limited to both buffer and the flag fitting in one segment, 65 536 bytes. The size parameter is a byte count of the size of the buffers. Each buffer is equal in size and must be of size “Size.” If the buffers are not equal in size, then cx should be set equal to the size of the smaller buffer.

Buffer_ID–AH—The Buffer_ID is a way for the application program to access and identify this particular error buffer at a later point. If successful in installing this buffer, the TSR will return along with the status, an identification number. This number is a binary value used to remove these buffers.

5.14 Remove Receive Error Buffer—This function removes the receive error buffer from the TSR; it releases the memory assignment of the receive error buffer and it releases the buffers back to the application program. The Buffer_ID that was returned by the TSR in Sub-Function 12, "Install Receiver Error Buffer," must be specified.

Call with:

```
AH =  1   =                          Function:  Communications
AL =  13  =                  Sub-Function:  Remove Receive Error Buffer
CL =          Buffer Identification Number
```

Returns with:

```
AL =  0   =  Status_OK
      1   =  Function value out of range.
      2   =  Sub-function out of range.
      3   =  This function is not supported.
      5   =  Invalid ID, is not found in current list.
      7   =  No buffer has been allocated.
```

Coding Examples:

Assembly Language:

```
Err_ID  DB ?                        ; TSR assigned ID number for this buffer
MOV CL,Err_ID                       ; Load buffer identification number
MOV AX,010DH                        ; Function/Sub-function = Remove Error Buffer
INT 62H                             ; Gateway to TSR interface
; Error checking                    ;
```

C Language:

```
#include   <dos.h>
union REGS inregs, outregs;
#define sErr_Buffer 200             /*  Size in bytes of buffers                    */
union REGS inregs, outregs;
struct SREGS segregs;
struct Err_Buffer
{
unsigned int   Flag                 /*  Control/Status Flag                         */
unsigned charBuffer1[sErr_Buffer],  /*  Define 2 buffers for                        */
              Buffer2[sErr_Buffer]; /*  TSR to switch between                       */
unsigned char ID;                   /*  Storage of TSR/assigned number of buffer    */
};
struct Err_Buffer Receiver;
inregs.h.cl = Receiver.ID;          /*  Buffer's identification number              */
inregs.x.ax = 0x010d;               /*  Function/Sub-function = Remove Receive Error*/
                                    /*  Buffer                                      */
int86x(0x62, &inregs, &outregs);    /*  Gateway to TSR interface                    */
/* Error checking of return status  */
}
```

5.15 Install Timed Transmit Buffer—This function takes a buffer containing timer stamps and complete packets and transmits the packets out over the network at the time specified by the timer stamps. The purpose of this command is to allow the application program to transmit a large buffer of data on a timed basis. A file can be built from the packets captured off the bus via the Receive Filter command, Function 1, Sub-Function 6, or it can be created artificially. Once loaded into a section of memory, this function can take that data and transmit it out, simulating what was received earlier in a timed fashion.

The format of this buffer is identical to that of the receive command and, thus, can be read from disk and loaded directly into the TSR. Transmission are triggered by the timer stamp in front of each packet, in 1 ms increments. This transmit section does not operate as the other transmitter. It will start transmitting after a period of Start_Out loaded into register CX.

Byte count, shown as follows, represents the size of the status, reserved, timer, and data fields. It includes the CHECKSUM in its size, but not itself. End of data is signified by a NULL character or zero in the Size position. These packets must be complete packets including MIDs, PIDs, Data, and CHECKSUM.

Call with:

AH	=	1	= Function: Communications
AL	=	14	= Sub-Function: Install Transmit Buffer
CX	=		Delay Timer before starting output
ES	=		Segment where Buffer and Flag Reside
BX	=		Address of the Control/Status Flag
DI	=		Start Address of the Buffer

Returns with:

AL	=	0	= Status_OK
		1	= Function value out of range
		2	= Sub-function out of range
		3	= This function not supported
		4	= Insufficient room in TSR's buffer space.
		7	= No buffer has been allocated.
		9	= Invalid parameter list.

Coding Examples:

Assembly Language:

```
sTimed_Tx_Buffer      EQU  1000          ; buffers maximum size
Timed_Tx_Command      EQU  0005H         ; command word to TSR
Transmit_Delay        EQU  1000          ; 1 s start up delay
Timed_Tx_Buffer       DB   sTimed_Tx_Buffer Dup(?)
Timed_Tx_Flag         DW   ?
;
; Buffer will need to be loaded.
;
MOV Timed_Tx_Flag,Timed_Tx_Command ; Set control flag
MOV AX,Seg Timed_Tx_Buffer         ; segment address of buffer and
MOV ES,AX                          ; Flag into Extra Segment
MOV BX,OffSet Timed_Tx_Flag        ; Offset address of control flag
MOV DI,OffSet Timed_Tx_Buffer      ; Offset address of buffer
MOV CX,Transmit_Delay              ; Start up delay period
MOV AX,010EH                       ; Function/Sub-Function =
                                   ; Install Timed Transmit Buffer
```

```
        INT 62H                                 ; Gateway to TSR interface
                                                ;
        ; Error checking here                   ;
```

C Language:

```
#include   <dos.h>
#define    sTimed_Tx_Buffer     10000      /**   buffers maximum size                **/
#define    Timed_Tx_Command     0x0005     /**   command word to TSR                 **/
#define    Transmit_Delay       1000       /**   1 s start up delay                  **/
FILE        *Packet_Stream
union      REGS      inregs, outregs;
struct     SREGS     segregs;
unsigned   char      Timed_Tx_Buffer[sTimed_Tx_Buffer];
unsigned   int       Timer_Tx_Flag;
unsigned   int       Number_Read;
/**
 ** Open the file called “data” in our current directory path for reading          **
 ** binary form.                                                                   **
 **/
if ((Packet_Stream = fopen(“data”, “r+b”)) ! = NULL)
{
/**
 ** Read data from file and store it in buffer “Timed_Tx_Buffer.”  Close the file, load   **
 ** command word into control flag, then load registers with segment and offset of this  **
 ** buffer and control flag.  CX gets the delay period for starting transmission of the  **
 ** buffers contents.  Be sure last packet size is zero.  Set Function & Sub-Function to **
 ** “Install Timed Transmit Buffer” and do a cause interrupt to access TSR application    **
 ** interface.                                                                     **/

Number_Read = fread((char*) Timed_Tx_Buffer, sizeof(char), sTimed_Tx_Buffer – 2,
Packet_Stream);
fclose(Packet_Stream);
if (Number_Read < (sTimed_Tx_Buffer – 2))
{
Timed_Tx_Flag = Timed_Tx_Command;
segregs.es = ((unsigned)((long)(void_far *)(Timed_Tx_Buffer) > > 16));
inregs.x.di = ((unsigned)((void_far *)(Timed_Tx_Buffer)));
inregs.x.bx = ((unsigned)((void_far*)(Timed_Tx_Flag)));
Timed_Tx_Buffer[Number_Read] = 0;
Timed_Tx_Buffer[Number_Read + 1] = 0;
inregs.x.cx = Transmit_Delay;
inregs.x.ax = 0x010e;
int86x(0x62, &inregs, &outregs, &segregs);
/**
 ** Error checking of return status                                               **/
}
```

NOTE—

Segment, ES—This is the segment register in the application program where the buffer and flag reside.

Buffer, DI—Offset address to the transmit buffer residing in the application program's memory. Data in this buffer follows the format shown in the receive filters. The data should not be altered or changed once it has been loaded by this command.