X3T9.2/91-101

NOVELL

12 July 1991

John Lohmeyer
NCR Corporation
3718 N. Rock Road
Wichita, KS 67226

Dear John,

I have forwarded for committee consideration copies of the proposed Standard AT Compatible Register Map and the Common Configuration Method for storage devices. These two documents outline a standard way of addressing AT compatible controllers/host bus adapters as well as a standard way of storing device specific geometry and information on ATA/IDE storage devices.

The proposed Standard AT Compatible Register Map defines four "Standard" sets of addresses based at; 1F0h, 170h, 1E8h, and 168h.

The device-specific information is recorded on the device in logical sector number two. Any driver or operating system can access the information for configuration purposes, etc. Specific fields can be updated or maintained by the operating software if desired. If this were done, a new check sum would be recomputed and saved in the location defined. The device-specific information would either be recorded on the device by the manufacturer or by a separate utility program.

Using the Standard Register Map and the Common Configuration Method, a stardard software driver would be able to locate each controller/ host bus adapter and identify each storage device with its specific information.

This would provide a standard way to access disk drive geometry in configurations using multiple controllers/host bus adapters and multiple storage devices attached to each controller/host bus adapter.

This proposal is not specific to any operating system but is intended for general industry use.

Sincerely,

Royes B Richins

Royes B. Richins
Sr. Software Engineer

cc Larry Lamers
Maxtor Corp.

NOVELL, INC.
122 EAST
1700 SOUTH
PROVO, UTAH
84606
801-429-7000
800-453-1267

Proposed Common Configuration Method

July 9, 1991

The Common Configuration Method (CCM) simplifies configuration of disks and other storage devices. This definition includes a Standard AT Compatible Register Map, and a configuration sector layout. Drivers incorporating this method of configuration will be able to find the controller and recognize the geometry and options of any media formatted and containing the CCM signature and required information in the indicated fields in sector 02 (third logical sector of the media on the device, assuming 512 byte sectors). This method of configuring for drives allow media to be identified universally independent of a given operating system or system BIOS. A driver incorporating this technique for configuration will be required to read sector 02 in the driver, recognize the Common Configuration Signature, and get the necessary information from the fields as defined in the Common Configuration sector layout in the following pages.

Common Configuration Method
Sector 02 Layout

Field	Offset(hex)	Offset(dec)	Data Type	Description
1	000 - 0FF	000 - 255	unsigned char [256]	**Vendor Unique** Reserved for vendor use
2*	100 - 101	256 - 257	unsigned int	**Signature** Contains the signature (55AAh)
3	102 - 105	258 - 261	unsigned long	**User blocks (low-order long word)** Low order half of following
4	106 - 109	262 - 265	unsigned long	**User blocks (high-order long word)** The user sectors divided by the block size**
5*	10A - 10B	266 - 267	unsigned int	**User Data Heads** Number of user-accessible data heads**
6*	10C - 10F	268 - 271	unsigned long	**User Cylinders** Number of user accessible cylinders** (not including spares)
7*	110 - 111	272 - 273	unsigned int	**Average Sectors Per Track** Average number of user-accessible sectors in each track**
8*	112 - 115	274 - 277	unsigned long	**User Sectors (low-order long word)** lower half of number of sectors (below)
9*	116 - 119	278 - 281	unsigned long	**User Sectors (high-order long word)** Total user accessible sectors on device**
10	11A - 11B	282 - 283	unsigned int	**BlockSize** Data transfer size in sectors per block**
11*	11C - 11D	284 - 285	unsigned int	**Sector Data Length** Number of data bytes in each sector**
12	11E - 13D	286 - 317	unsigned char [32]	**Support Field** Feature identification (for vendor use)
13*	13E - 13F	318 - 319	unsigned int	**Controller Interface Type** 0 = unknown 3 = ESDI 6 = ST-506 1 = IDE/ATA 4 = SMD 2 = SCSI 5 = IPI If greater than 6, refer to Controller Name
14*	140 - 14F	320 - 335	unsigned char [16]	**Model Name***
15	150 - 15F	336 - 351	unsigned char [16]	**Controller Name***

Common Configuration Method
Sector 02 Layout (cont)

Field	Offset(hex)	Offset(dec)	Data Type	Description
16*	160 - 161	352 - 353	unsigned int	**Peripheral Device Type****** 00 = Hard disk 01 = Magnetic Tape 02 = Printer 03 = Processor Device 04 = WORM Device 05 = CD-ROM 06 = Scanner 07 = Optical Disk 08 = Medium Changer Device 09 = Communications Device 0Ah-0Bh = Defined by ASC IT8 0Ch-1Eh = Reserved 1Fh = Unknown or no device type 7Fh = Logical Unit Missing 80h-FFh = Vendor Unique
17*	162 - 175	354 - 373	unsigned char [20]	**Device Serial Number*****
18	176 - 179	374 - 377	unsigned char [4]	**Unique Device Address** In family
19	17A - 199	378 - 409	unsigned long [8]	**Start-up Sector Pointers** Sector numbers for devices that require special start-up routines
20	19A - 1FB	410 - 507	unsigned char [98]	**Reserved** Reserved for future use
21*	1FC - 1FF	508 - 511	unsigned long	**CRCDoubleWord** 32-bit CRC of the CCM area (0100h-01FBh)

* Fields that must contain valid data if the signature is written to bytes 100h-101h
** An absolute count; i.e., a whole number total; 0=zero units, 1=one unit, etc.
*** A null-terminated and null-padded ASCII string
**** From SCSI ANSI Standard X3.131-1986, 23 Jun 1986, p. 70; see also SCSI II ANSI Standard X3.131-1990, 31 Aut 1990, p. 7-21
Note: The decimal whole numbers in brackets [] indicate the storage allocation in bytes in the designated data type

DEFINITIONS

byte An 8-bit unit, such as a character

word A 16-bit unit, such as an integer

long word A 32-bit unit, such as a long integer

vendor Manufacturer of the device

This section details the contents and definitions of all the fields specified by the Common Configuration Method (CCM) for sector 02 of peripheral devices. All addresses are designated in hex and pertain to the data field of sector 02. All fields are unsigned.

Field 1: **Vendor Unique** 256 bytes unsigned char

The **Vendor Unique** field is reserved for purposes specific to the manufacturer and is comprised of addresses 000h through 0FFh (the lower-addressed half of sector 02); Because the information stored at this location is vendor-specific, the generic data type unsigned char is assigned to it.

Field 2: **Signature** 2 bytes unsigned int

The Signature field contains the hex number 55AA (decimal 21930) at address 100h of sector 02 to indicate the remaining data follows the CCM standard. This signature also indicates that valid data is stored in the User Data Heads, Average Sectors Per Track, User Type, and Serial Number fields, and a valid checksum for the CCM data is stored in the Checksum field.

Field 3: **User Blocks**, low order long word 4 bytes unsigned long

Field 4: **User Blocks**, high order long word 4 bytes unsigned long

The lower 32 bits of the 64-bit number representing the number of blocks on the device is stored at address 102h, the low order long word of the User Blocks field. The upper 32 bits of the same 64-bit number is stored at address 106h, the high order long word of the User Blocks field. This is equal to the User Sectors divided by the Block Size. The number of user blocks is a whole number, indicating an actual count of the number of blocks available to the user; i.e., 0 = zero blocks, 1 = one block, 2 = two blocks, etc.

Field 5: **User Data Heads** 2 bytes unsigned int

The number of data heads on the device accessible to the user is stored in the User Data Heads field at address 10Ah and does not include servo heads. Valid data must be present in this field if the proper signature is written to the Signature field. The number of user data heads is a whole number, indicating an actual count of the number of data heads available to the user; i.e., 0 = zero heads, 1 = one head, 2 = two heads, etc.

Field 6: **User Cylinders** 4 bytes unsigned long

The number of cylinder so the device accessible to the user is stored in the User Cylinders field at address 10Ch and does not include spare cylinders. Valid data must be present in this field if the proper signature is written to the Signature field. The number of user cylinders is a whole number, indicating an actual count of the number of cylinders available to the user; i.e., 0 = zero cylinders, 1 = one cylinder, 2 = two cylinders, etc.

Field 7: **Average Sectors Per Track** 2 bytes unsigned int

The average number of sectors in each track accessible to the user is stored in the Average Sectors Per Track field at address 110h. Valid data must be present in this field if the proper signature is written to the Signature ;field. The average number of sectors per track is a whole number, indicating an actual count of the average number of sectors per track available to the user; i.e., 0 = zero sectors per track, 1 = one sector per track, 2 = two sectors per track, etc.

Field 8: **User Sectors**, low order long word 4 bytes unsigned long

Field 9: **User Sectors**, high order long word 4 bytes unsigned long

The lower 32 bits of the 64-bit number representing the number of sectors on the device is stored at address 112h, the low order long word of the User Sectors field. The upper 32 bits of the same 64-bit number is stored at address 116h, the high order long word of the User Sectors field. This is equal to the User Blocks multiplied by the Block Size. Valid data must be present in this field if the proper signature is written to the Signature field. The number of user sectors is a whole number, indicating an actual count of the number of sectors available to the user; i.e., 0 = zero sectors, 1 = one sector, 2 = two sectors, etc.

Field 10: **Block Size** 2 bytes unsigned int

The data transfer size in the number of sectors per block is stored in the Block Size field at address 11Ah. The block size is a whole number, indicating an actual count of the number of sectors per block available to the user; i.e., 0 = zero sectors per block, 1 = one sector per block, 2 = two sectors per block. etc.

Field 11: **Sector Data Length** 2 bytes unsigned int

The number of data bytes in each sector is stored in the Sector Data Length field at address 11Ch. Valid data must be present in this field if the proper signature is written to the Signature field. The sector data length is a whole number, indicating an actual count of the number of data bytes per sector available to the user; i.e., 0 = zero data bytes per sector, 1 = one data byte per sector, 2 = two data bytes per sector, etc.

Field 12: **Support Field** 32 bytes unsigned char

The support Field is an information field that is allocated for feature identification of the device and is reserved for use by the vendor.

Field 13: **Controller Interface Type** 2 bytes unsigned int

The 16-bit number that identifies the type of interface the device communicates with is stored in the Controller Interface Type field at address 13Eh. This information is defined as follows:

0 = unknown controller type	3 = ESDI	6 = ST-506
1 = IDE/ATA	4 = SMD	
2 = SCSI I/SCSI II	5 = IPI	

Controller types represented by numbers greater than 6 are identified in the Controller Name field. Valid data must be present in this field if the proper signature is written to the Signature field.

Field 14: **Model Name** 16 bytes unsigned char

The model name of the device is stored in the Model Name field at address 140h. This is a null-padded and null-terminated ASCII string, meaning that (1) the last byte must be a null (00h) character and (2) all other unused bytes must also be null characters.

Field 15: **Controller Name** 16 bytes unsigned char

The controller name of the device is stored in the Controller Name field at address 150h. This is a null-padded and null-terminated ASCII string, meaning that (1) the last byte must be a null (00h) character and (2) all other unused bytes must also be null characters.

Field 16: **Peripheral Device Type** 2 bytes unsigned int

The 16-bit number that identifies the medium or peripheral device type is stored in the Peripheral Device Type field at address 160h. This information is defined as follows:

00h	= Hard Disk	08h	= Medium Changer Device
01h	= Magnetic Tape	09h	= Communications Device
02h	= Printer	0Ah-0Bh	= Defined by ASC IT8
03h	= Processor Device	0Ch-1Eh	= Reserved
04h	= WORM Device	1Fh	= Unknown/No Device Type
05h	= CD-ROM	7Fh	= Logical Unit Missing
06h	= Scanner	80h-FFh	= Vendor Specific
07h	= Optical Disk		

Valid data must be present in this field if the proper signature is written to the Signature field.

Field 17: **Serial Number** 20 bytes unsigned char

The serial number of the device is stored as an ASCII string in the Serial Number field at address 162h. This is a null-padded and null-terminated ASCII string, meaning that (1) the last byte must be a null (00h) character and (2) all other unused bytes must also be null characters. Valid data must be present in this field if the proper signature is written to the Signature field.

Field 18: **Unique Device Address** 4 bytes unsigned char

The unique address that distinguishes the device from other members of the same family is stored in the Unique Device Address field at address 176h.

Field 19: **Startup Sector Pointers** 32 bytes unsigned long

The Startup Sector Pointers field is provided to store sector numbers that locate routines required by some devices to complete the startup process.

Field 20: **Reserved** 98 bytes unsigned char

The Reserved field is allocated for future expansion.

Field 21: **CRCDoubleWord** 4 bytes unsigned long

This field contains a 32-bit CRC of the CCM data starting at location 0100h and ending with location 01FBh. The CRC is calculated using the supplied "C" routine. Please note that the CRC accumulator is initialized to all ones (0FFFFFFFFh) and that the resulting value is inverted. This standard technique prevents generating a matching CRC when data is erroneously shifted by one or more bytes, with more or less initial zero bytes. The polynomial used in the 32-bit CRC is found in the ANSI X3.66 specification, or FED-STD-1003, and is represented as:

$$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X^{1}+X^{0}$$

Valid data should be present in this field if the proper signature is written to the **Signature** field.

Following are sample routines to calculate the CRCDoubleWord in both "C" and Assembler, along with a sample definition file to allow the 32-bit mode assembler routine to be loaded as an NLM, exporting the routine for drivers and other NLMs:

Sample 32-bit CRC32 Routine

```
 COMMENT ~
======================================================================

*ENTRYPOINT:     GenerateCRC32

*PURPOSE:        Generates 32-bit CRCs for data blocks (32-bit mode flat model)

*PROTOTYPE:      CRC = GenerateCRC32(bufferlength, @buffer)

*PARAMETERS:     bufferlength     -   LONG buffer length in bytes

                 @buffer          -   LONG pointer to data buffer

*RETURNS:        CRC              -   LONG new accumulated CRC (EAX)

*NOTES:          This CRC polynomial is in the ANSI X3.66 specification.  32-bit CRCs
                 are approximately 100,000 times more likely to detect an error than
                 16-bit CRCs.  The 32-bit polynomial used is:

                  32  26  23  22  16  12  11  10  8  7  5  4  2  1  0
                 X  +X  +X  +X  +X  +X  +X  +X  +X +X +X +X +X +X +X

======================================================================
*:~
    name     GenerateCRC32
    assume   ds: OSDATA, es: OSDATA, ss: OSDATA

    CPush    macro
    push     ebp
    push     ebx
    push     esi
    push     edi
    cld
    endm

    CPop     macro
    pop edi
    pop esi
    pop ebx
    pop ebp
    endm
```

```
; following used to fetch values from the stack
ParmOffset   equ 20      ;for ebx,ebp,esi,edi plus near call
Parm0        equ ParmOffset + 0
Parm1        equ ParmOffset + 4
poly         equ 04C11DB3h

OSDATA   segment rw public 'DATA'
OSDATA   ends

OSCODE   segment er public 'CODE'
         assume   cs: OSCODE
         public   GenerateCRC32

         align    4
CRC32C   proc
GenerateCRC32    label    near

    CPush                     ;push standard registers for C
    mov ecx, [esp + Parm0]    ;*** get # bytes to process
    mov esi, [esp + Parm1]    ;*** get buffer pointer
    xor eax, eax              ;initialize crc accumulator
    dec eax                   ;to all ones
    mov edx, poly             ;get polynomial constant
    jecxz   exit              ;*** if no bytes to process
    align   4
byteloop:
    rol eax, 8
    xor al, [esi]             ;update with new byte
    ror eax, 8
    mov bh, 8                 ;set bit counter
    align   4
bitloop:
    shl eax, 1                ;shift accumulator left 1 bit
    jnc bitzero               ;if bit was zero
    xor eax, edx              ;update accumulator
    align   4
bitzero:
    dec bh                    ;done with byte ?
    jnz bitloop               ;no - do next bit
    inc esi                   ;adjust buffer pointer
    loop    byteloop          ;loop til buffer processed
exit:
    xor eax,-1                ;1's complement result
    CPop                      ;restore saved regs
    ret                       ;return to caller

CRC32C   endp
OSCODE   ends
    end
```

Sample DOS real-mode CRC32 C-callable routine

```
 COMMENT ~
======================================================================

*MODULE:     CRC32.ASM

*PURPOSE:    Generates 32-bit CRCs for data blocks (DOS Real-Mode)

*ENVIRONMENT:           DOS Client      *MODIFIES ENVIRONMENT:  n
*MODEL:                 HUGE            *BLOCKING:              n
*MODIFIES INTERRUPTS:   n               *RESTORES INTERRUPTS:   n
*REQ CALL DISABLED:     n               *REQ CALL ENABLED:      y
*C-CALLABLE:            y               *CALLABLE FROM INT:     y

*PROTOTYPE:  CRC = CRC32(bufferlength, @buffer)

*PARAMETERS:     bufferlength     -   LONG buffer length in bytes

                 @buffer          -   LONG pointer to data buffer

*RETURNS:        CRC              -   (AX:DX) LONG new accumulated CRC

*MODIFIES:       AX, BX, CX, DX, FLAGS

*NOTES:          Assemble with the /MX option.

        The 32-bit polynomial used is:

         32  26  23  22  16  12  11  10  8  7  5  4  2  1  0
        X  +X  +X  +X  +X  +X  +X  +X  +X +X +X +X +X +X +X

        The polynomial is defined in ANSI X3.66.

======================================================================
*:~
Parms    equ 6            ;far call plus bp
Parm0Lo  equ Parms+0
Parm0Hi  equ Parms+2
Parm1Lo  equ Parms+4
Parm1Hi  equ Parms+6
polylo   equ 1DB3h        ;low word of 32-bit polynomial
polyhi   equ 04C1h        ;high word - note that 32 bit polynomials actually are
                          ;33-bit by definition
```

Sample DOS real-mode CRC32 C-callable routine (cont)

```
    public   _CRC32
_TEXT   segment DWORD public 'CODE'
    assume  CS:_TEXT
GenerateCRC32    proc    far
_CRC32  label   DWORD
; this routine alters AX, BX, CX, DX (Turbo C or MSC 5.1 compatible)
    push    bp
    mov bp, sp                ;setup bp
    push    es                ;save
    push    si                ;save
    push    di                ;save
    mov cx, [bp + Parm0Lo]    ;*** get # bytes to process
    mov bx, [bp + Parm0Hi]    ;*** get high word of count
    mov si, [bp + Parm1Lo]    ;*** get buffer pointer offset
    mov ax, [bp + Parm1Hi]    ;*** get segment address
    mov es, ax                ;set up ES
    mov ax, 0FFFFh            ;initialize 32-bit accumulator
    mov dx, ax
    or  bx, bx                ;hi-order count non-zero
    jnz SHORT byteloop        ;yes - do it
    jcxz    SHORT exit        ;if no bytes to process
    align   2
byteloop:
    xor dh, ES:[si]           ;update with new byte
    mov bh, 8                 ;loop for each bit in byte
    align   2
bitloop:
    shl ax,1                  ;shift accumulator left 1 bit
    rcl dx,1                  ;all 32 bits
    jnc SHORT bitzero         ;if bit shifted out was zero
    xor ax, polylo            ;update crc with polynomial
    xor dx, polyhi
    align   2
bitzero:
    dec bh                    ;done with byte ?
    jnz bitloop               ;no - do next bit
    inc si                    ;adjust buffer pointer
    jz  SHORT fixseg          ;skip if at segment end

    align   2
bytechk:
    loop    byteloop          ;loop til buffer processed
    dec bl                    ;theoretical limit 16M bytes
    jnz byteloop              ;until all done
exit:
    xor ax, 0FFFFh            ;1's complement result
    xor dx, 0FFFFh            ;per ANSI standard
    pop di                    ;restore
    pop si                    ;restore
    pop es                    ;restore
    pop bp                    ;restore
    ret                       ;return to caller
fixseg:
    mov si, es                ;fixup ES:si if end of segment reached
    add si, 1000h             ;move up 64k
    mov es, si                ;restore
    xor si, si                ;registers
    jmp bytechk               ;return to mainline

GenerateCRC32    endp
_TEXT   ends
    end
```

Sample CRC32 routine in C

```
/* CRC32.C  Generates 32-bit CRCs for data blocks                          */
/*                                                                         */
/*    crc32 = CGenerateCRC32(bufferlength, @buffer)                        */
/*                                                                         */
/* PARAMETERS:      bufferlength     -   unsigned long buffer length       */
/*                                                                         */
/*                  @buffer          -   pointer to data buffer            */
/*                                                                         */
/* RETURNS:         crc32            -   unsigned long 32-bit CRC          */
/*                                                                         */
/* NOTES:           The 32-bit polynomial used is:                         */
/*                                                                         */
/*       32  26  23  22  16  12  11  10  8  7  5  4  2  1  0               */
/*      X  +X  +X  +X  +X  +X  +X  +X  +X +X +X +X +X +X +X                */
/*                                                                         */
/*      The polynomial is defined in ANSI X3.66.                           */
/*                                                                         */

unsigned long CGenerateCRC32(unsigned long bufferlength,
            unsigned char *buffer)
{
    unsigned long    crc;
    unsigned long    tmp;
    unsigned long    bytecnt;
    unsigned long    bitcnt;

    crc = 0xffffffffL;
    if (bufferlength == 0) return (crc ^ 0xffffffffL);
    for (bytecnt = 0, bytecnt < bufferlength, bytecnt++)
        {
        tmp = (unsigned long int) *buffer;
        crc ^= (tmp << 24);
        for (bitcnt = 0, bitcnt < 8, bitcnt++)
            {
            tmp = (crc & 0x80000000L);
            crc = (crc << 1);
            if (tmp) crc ^= 0x04C11DB3L;
            }
            buffer++;
        }
    return (crc ^ 0xffffffffL);
}
```

Proposed Standard AT Compatible

Register Map

25 Jun 1991

This document lists the proposed Standard AT Compatible Register Map for IBM AT systems and true compatibles.

Proposed Standard AT Compatible Register Map

(all addresses are in hex)

Register Name	Primary Address	Secondary Address	Tertiary Address	Quaternary Address	Register Type
Data	1F0	170	1E8	168	Read/Write
Error	1F1	171	1E9	169	Read Only
Write Precomp	1F1	171	1E9	169	Write Only
Sector Count	1F2	172	1EA	16A	Write Only
Sector Number	1F3	173	1EB	16B	Read/Write
Cylinder Low	1F4	174	1EC	16C	Read/Write
Cylinder High	1F5	175	1ED	16D	Read/Write
Drive/Head	1F6	176	1EE	16E	Read/Write
Status	1F7	177	1EF	16F	Read Only
Command	1F7	177	1EF	16F	Write Only
Alternate Status	3F6	376	3EE	36E	Read Only
Fixed Disk	3F6	376	3EE	36E	Write Only
Digital Input	3F7	377	3EF	36F	Read Only