expansion

COLLABORATORS

	TITLE : expansion		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 28, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

expansion

1.1 expansion.doc

1.2 expansion.library/AddDosNode

```
NAME
    AddDosNode -- mount a disk to the system

SYNOPSIS
    ok = AddDosNode( bootPri, flags, deviceNode )
    D0               D0       D1     A0

FUNCTION
    This routine makes sure that your disk device (or a device
    that wants to be treated as if it was a disk...) will be
    entered into the system.  If the dos is already up and
    running, then it will be entered immediately.  If the dos
    has not yet been run then the data will be recorded, and the
    dos will get it later.

    We hope to eventually try and boot off a disk device.  We will
    try and boot off of each device in turn, based on priority,
    iff there is no boot floppy in the floppy disk drive.  As of
    this writing that facility does not yet exist.

    There is only one additional piece of magic done by AddDosNode.
    If there is no executable code specified in the deviceNode
    structure (e.g. dn_SegList, dn_Handler, and dn_Task are all
    null) then the standard dos file handler is used for your
    device.
    Documentation note: a "task" as used here is a dos-task, not
    an exec-task.  A dos-task, in the strictest sense, is the
    address of an exec-style message port.  In general, it is
    a pointer to a process's pr_MsgPort field (e.g. a constant
    number of bytes after an exec port).

INPUTS
    bootPri -- a BYTE quantity with the boot priority for this disk.
        This priority is only for which disks should be looked at:
        the actual disk booted from will be the first disk with
        a valid boot block.  If no disk is found then the "bootme"
        hand will come up and the bootstrap code will wait for
        a floppy to be inserted.  Recommend priority assignments are:

            +5 -- unit zero for the floppy disk.  The floppy should
                always be highest priority to allow the user to
                abort out of a hard disk boot.
            0 -- the run of the mill hard disk
            -5 -- a "network" disk (local disks should take priority).
            -128 -- don't even bother to boot from this device.

    flags -- additional flag bits for the call:
        ADN_STARTPROC (bit 0) -- start a handler process immediately.
            Normally the process is started only when the device node
            is first referenced.  This bit is meaningless if you
            have already specified a handler process (non-null dn_Task).

    deviceNode -- a legal DOS device node, properly initialized.
        Typically this will be the result of a MakeDosNode()
        call, but feel free to manufacture your own if you need
        to.  If deviceNode is null then AddDosNode does nothing.

RESULTS
    ok - non-zero everything went ok, zero if we ran out of memory
        or some other weirdness happened.

EXAMPLES
    /* enter a bootable disk into the system.  Start a file handler
    ** process immediately.
    */
    AddDosNode( 0, ADNF_STARTPROC, MakeDosNode( paramPacket ) );

BUGS
    The flexible boot strategy is only that -- strategy.  It still
    needs to be reflected in code somewhere.

SEE ALSO
    MakeDosNode

BUGS
```

1.3 expansion.library/MakeDosNode

```
NAME
    MakeDosNode -- construct dos data structures that a disk needs

SYNOPSIS
    deviceNode = MakeDosNode( parameterPkt )
    D0                        A0

FUNCTION
    This routine manufactures the data structures needed to enter
    a dos disk device into the system.  This consists of a DeviceNode,
    a FileSysStartupMsg, a disk environment vector, and up to two
    bcpl strings.  See the libraries/dosextens and libraries/filehandler
    include files for more information.

    MakeDosNode will allocate all the memory it needs, and then
    link the various structure together.  It will make sure all
    the structures are long-word aligned (as required by the DOS).
    It then returns the information to the user so he can
    change anything else that needs changing.  Typically he will
    then call AddDosNode() to enter the new device into the dos
    tables.

INPUTS
    parameterPkt - a longword array containing all the information
        needed to initialize the data structures.  Normally I
        would have provided a structure for this, but the variable
        length of the packet caused problems.  The two strings are
        null terminated strings, like all other exec strings.

        longword    description
        --------    -----------
        0           string with dos handler name
        1           string with exec device name
        2           unit number (for OpenDevice)
        3           flags (for OpenDevice)
        4           # of longwords in rest of enviroment
        5-n         file handler environment (see libraries/filehandler.h)

RESULTS
    deviceNode - pointer to initialize device node structure, or
        null if there was not enough memory.

EXAMPLES
    /* set up a 3.5" amiga format floppy drive for unit 1 */

    char execName[] = "trackdisk.device";
    char dosName[] = "df1";

    ULONG parmPkt[] = {
        (ULONG) dosName,
        (ULONG) execName,
        1,                  /* unit number */
        0,                  /* OpenDevice flags */

        /* here is the environment block */
        11,                 /* table upper bound */
        512>>2,             /* # longwords in a block */
        0,                  /* sector origin -- unused */
        2,                  /* number of surfaces */
        1,                  /* secs per logical block -- unused */
        11,                 /* secs per track */
        2,                  /* reserved blocks -- 2 boot blocks */
        0,                  /* ?? -- unused */
        0,                  /* interleave */
        0,                  /* lower cylinder */
        79,                 /* upper cylinder */
        5,                  /* number of buffers */
    };

    struct Device Node *node, *MakeDosNode();

    node = MakeDosNode( parmPkt );

BUGS

SEE ALSO
    AddDosNode
```

1.4 expansion.library/AddConfigDev

```
NAME
    AddConfigDev - add a new ConfigDev structure to the system

SYNOPSIS
    AddConfigDev( configDev )
                  A0

FUNCTION
    This routine adds the specified ConfigDev structure to the
    list of Configuration Devices in the system.

INPUTS
    configDev - a valid ConfigDev structure.

RESULTS

EXCEPTIONS

SEE ALSO
    RemConfigDev

BUGS
```

1.5 expansion.library/AllocBoardMem

```
NAME
    AllocBoardMem - allocate standard device expansion memory
```

```
SYNOPSIS
    startSlot = AllocBoardMem( slotSpec )
    D0                         D0

FUNCTION
    This function allocates numslots of expansion space (each slot
    is E_SLOTSIZE bytes).  It returns the slot number of the
    start of the expansion memory.  The EC_MEMADDR macro may be
    used to convert this to a memory address.

    AllocBoardMem() knows about the intracacies of expansion
    board hardware and will allocate the proper expansion
    memory for each board type.

INPUTS
    slotSpec - the memory size field of the Type byte of
            an expansion board

RESULTS
    startSlot - the slot number that was allocated, or -1 for error.

EXAMPLES
            struct ExpansionRom *er;
            slot = AllocBoardMem( er->er_Type & ERT_MEMMASK )

EXCEPTIONS
    Not typically called by user code.

SEE ALSO
    AllocExpansionMem, FreeExpansionMem, FreeBoardMem

BUGS
```

1.6 expansion.library/AllocConfigDev

```
NAME
    AllocConfigDev - allocate a ConfigDev structure

SYNOPSIS
    configDev = AllocConfigDev()
    D0

FUNCTION
    This routine returns the address of a ConfigDev structure.
    It is provided so new fields can be added to the structure
    without breaking old, existing code.  The structure is cleared
    when it is returned to the user.

INPUTS

RESULTS
    configDev - either a valid ConfigDev structure or NULL.

EXCEPTIONS
```

```
SEE ALSO
    FreeConfigDev

BUGS
```

1.7 expansion.library/AllocExpansionMem

```
NAME
    AllocExpansionMem - allocate expansion memory

SYNOPSIS
    startSlot = AllocExpansionMem( numSlots, slotOffset )
    D0                             D0        D1

FUNCTION
    This function allocates numslots of expansion space (each slot
    is E_SLOTSIZE bytes).  It returns the slot number of the
    start of the expansion memory.  The EC_MEMADDR macro may be
    used to convert this to a memory address.

    Boards that fit the expansion architecture have alignment
    rules.  Normally a board must be on a binary boundary of its
    size.  Four and Eight megabyte boards have special rules.
    User defined boards might have other special rules.

    The routine AllocBoardMem() knows about all the allocation
    rules for standard boards.  Most users will want to use
    that routine if they want memory for a standard expansion
    device.

    If AllocExpansionMem() succeeds, the startSlot will satisfy
    the following equation:

            (startSlot - slotOffset) MOD slotAlign = 0

INPUTS
    numSlots - the number of slots required.
    slotOffset - an offset from that boundary for startSlot.

RESULTS
    startSlot - the slot number that was allocated, or -1 for error.

EXAMPLES
            AllocExpansionMem( 2, 0 )

    Tries to allocate 2 slots on a two slot boundary.

            AllocExpansionMem( 64, 32 )

    This is the allocation rule for 4 meg boards.  It allocates
    4 megabytes (64 slots) on an odd 2 meg boundary.

EXCEPTIONS
    Not typically called by user code.
```

```
SEE ALSO
    FreeExpansionMem, AllocBoardMem, FreeBoardMem

BUGS
```

1.8 expansion.library/ConfigBoard

```
NAME
    ConfigBoard - configure a board

SYNOPSIS
    error = ConfigBoard( board, configDev )
    D0                   A0     A1

FUNCTION
    This routine configures an expansion board.  The board
    will generally live at E_EXPANSIONBASE, but the base is
    passed as a parameter to allow future compatibility.
    The configDev parameter must be a valid configDev that
    has already had ReadExpansionRom() called on it.

    ConfigBoard will allocate expansion memory and place
    the board at its new address.  It will update configDev
    accordingly.  If there is not enough expansion memory
    for this board then an error will be returned.

INPUTS
    board - the current address that the expansion board is
            responding.
    configDev - an initialized ConfigDev structure.

RESULTS
    error - non-zero if there was a problem configuring this board

EXCEPTIONS
    Not normally called by user code

SEE ALSO
    FreeConfigDev

BUGS
```

1.9 expansion.library/ConfigChain

```
NAME
    ConfigChain - configure the whole damn system

SYNOPSIS
    error = ConfigChain( baseAddr )
    D0                   A0

FUNCTION
```

```
    This is the big one!  This routine will take a base address
    (generally E_EXPANSIONBASE) and configure all the devices that
    live there.  This routine will call all the other routines
    that might need to be called.  All boards that are found will
    be linked into the configuration list.

INPUTS
    baseAddr - the base address to start looking for boards.

RESULTS
    error - non-zero if something went wrong.

EXCEPTIONS
    Not normally called by user code

SEE ALSO
    FreeConfigDev

BUGS
```

1.10 expansion.library/FindConfigDev

```
NAME
    FindConfigDev - find a matching ConfigDev entry

SYNOPSIS
    configDev = FindConfigDev( oldConfigDev, manufacturer, product )
    D0                         A0            D0            D1

FUNCTION
    This routine searches the list of existing ConfigDev
    structures in the system and looks for one that has
    the specified manufacturer and product codes.

    If the oldConfigDev is NULL the the search is from the
    start of the list of configuration devices.  If it is
    not null then it searches from the first configuration
    device entry AFTER oldConfigDev.

    A code of -1 is treated as a wildcard -- e.g. it matches
    any manufacturer (or product)

INPUTS
    oldConfigDev - a valid ConfigDev structure, or NULL to start
            from the start of the list.
    manufacturer - the manufacturer code being searched for, or
            -1 to ignore manufacturer numbers.
    product - the product code being searched for, or -1 to
            ignore product numbers.

RESULTS
    configDev - the next ConfigDev entry that matches the
            manufacturer and product codes, or NULL if there
            are no more matches.
```

```
EXCEPTIONS

EXAMPLES
    /* to find all configdevs of the proper type */
    struct ConfigDev *cd = NULL;

    while( cd = FindConfigDev( cd, MANUFACTURER, PRODUCT ) ) {
            /* do something with the returned ConfigDev */
    }

SEE ALSO

BUGS
```

1.11 expansion.library/FreeBoardMem

```
NAME
    FreeBoardMem - allocate standard device expansion memory

SYNOPSIS
    FreeBoardMem( startSlot, slotSpec )
                  D0         D1

FUNCTION
    This function frees numslots of expansion space (each slot
    is E_SLOTSIZE bytes).  It is the inverse function of
    AllocBoardMem().

INPUTS
    startSlot - a slot number in expansion space.
    slotSpec - the memory size field of the Type byte of
            an expansion board

RESULTS

EXAMPLES
    struct ExpansionRom *er;
    int startSlot;
    int slotSpec;

    slotSpec = er->er_Type & ERT_MEMMASK;
    startSlot = AllocBoardMem( er->er_Type & ERT_MEMMAK );

    if( startSlot != -1 ) {
        FreeBoardMem( startSlot, slotSpec );
    }

EXCEPTIONS
    If the caller tries to free a slot that is already in the
    free list, FreeBoardMem will Alert() (e.g. crash the
    system).

    Not normally called by user code

SEE ALSO
```

```
    AllocExpansionMem, FreeExpansionMem, AllocBoardMem

BUGS
```

1.12 expansion.library/FreeConfigDev

```
NAME
    FreeConfigDev - allocate a ConfigDev structure

SYNOPSIS
    FreeConfigDev( configDev )
                   A0

FUNCTION
    This routine frees a ConfigDev structure as returned by
    AllocConfigDev.

INPUTS
    configDev - a valid ConfigDev structure.

RESULTS

EXCEPTIONS

SEE ALSO
    AllocConfigDev

BUGS
```

1.13 expansion.library/FreeExpansionMem

```
NAME
    FreeExpansionMem - allocate standard device expansion memory

SYNOPSIS
    FreeExpansionMem( startSlot, numSlots )
                      D0         D1

FUNCTION
    This function allocates numslots of expansion space (each slot
    is E_SLOTSIZE bytes).  It is the inverse function of
    AllocExpansionMem().

INPUTS
    startSlot - the slot number that was allocated, or -1 for error.
    numSlots - the number of slots to be freed.

RESULTS

EXAMPLES

EXCEPTIONS
```

```
    If the caller tries to free a slot that is already in the
    free list, FreeExpansionMem will Alert() (e.g. crash the
    system).

    Not normally called by user code

SEE ALSO
    AllocExpansionMem, AllocBoardMem, FreeBoardMem

BUGS
```

1.14 expansion.library/GetCurrentBinding

```
NAME
    GetCurrentBinding - sets static board configuration area

SYNOPSIS
    actual = GetCurrentBinding( currentBinding, size )
                                A0              D0:16

FUNCTION
    This function writes the contents of the "currentBinding"
    structure out of a private place.  It may be set via
    SetCurrentBinding().  This is really a kludge, but it is
    the only way to pass extra arguments to a newly configured
    device.

    A CurrentBinding structure has the name of the currently
    loaded file, the product string that was associated with
    this driver, and a pointer to the head of a singly linked
    list of ConfigDev structures (linked through the cd_NextCD
    field).

    Many devices may not need this information; they have hard
    coded into themselves their manufacture number.  It is
    recommended that you at least check that you can deal with
    the product code in the linked ConfigDev structures.

INPUTS
    currentBinding - a pointer to a CurrentBinding structure

    size - the size of the user's binddriver structure.  No
        more than this much data will be copied.  If size is
        larger than the libraries idea a CurrentBinding size,
        then the structure will be null padded.

RESULTS
    actual - the true size of a CurrentBinding structure is returned.

EXAMPLES

EXCEPTIONS

SEE ALSO
    GetCurrentBinding
```

```
BUGS
```

1.15 expansion.library/ObtainConfigBinding

```
NAME
    ObtainConfigBinding - try to get permission to bind drivers

SYNOPSIS
    ObtainConfigBinding()

FUNCTION
    ObtainConfigBinding gives permission to bind drivers to
    ConfigDev structures.  It exists so two drivers at once
    do not try and own the same ConfigDev structure.  This
    call will block until it is safe proceed.

    Individual drivers to not need to call this routine.  It
    is intended for BindDriver program, and others like it.
    If your drivers won't be loaded via the standard method,
    you may need to lock out others.

    It is crucially important that people lock out others
    before loading new drivers.  Much of the data that is used
    to configure things is statically kept, and others need
    to be kept from using it.

    This call is build directly on Exec SignalSemaphore code
    (e.g. ObtainSemaphore).

INPUTS

RESULTS

EXCEPTIONS

SEE ALSO
    ReleaseConfigBinding

BUGS
```

1.16 expansion.library/ReadExpansionByte

```
NAME
    ReadExpansionByte - read a byte nybble by nybble.

SYNOPSIS
    byte = ReadExpansionByte( board, offset )
    D0                        A0     D0

FUNCTION
    ReadExpansionByte reads a byte from a new-style expansion
    board.  These boards have their readable data organized
    as a series of nybbles in memory.  This routine reads
    two nybbles and returns the byte value.

    In general, this routine will only be called by ReadExpansionRom.

    The offset is a byte offset into a ExpansionRom structure.
    The actual memory address read will be four times larger.
    The macros EROFFSET and ECOFFSET are provided to help get
    these offsets from C.

INPUTS
    board - a pointer to the base of a new style expansion board.
    offset - a logical offset from the board base

RESULTS
    byte - a byte of data from the expansion board, or -1 if there
            was an error reading from the board.

EXAMPLES
    byte = ReadExpansionByte( cd->BoardAddr, EROFFSET( er_Type ) );
    ints = ReadExpansionByte( cd->BoardAddr, ECOFFSET( ec_Interrupt ) );

EXCEPTIONS
    Not typically called by user code.

SEE ALSO
    WriteExpansionByte, ReadExpansionRom

BUGS
```

1.17 expansion.library/ReadExpansionRom

```
NAME
    ReadExpansionRom - read a boards configuration rom space

SYNOPSIS
    error = ReadExpansionRom( board, configDev )
    D0                        A0     A1

FUNCTION
    ReadExpansionRom reads a the rom portion of an expansion
    device in to cd_Rom portion of a ConfigDev structure.
    This routine knows how to detect whether or not there is
    actually a board there,

    In addition, the Rom portion of a new style expansion board
    is encoded in ones-complement format (except for the first
    two nybbles -- the er_Type field).  ReadExpansionRom knows
    about this and un-complements the appropriate fields.

INPUTS
    board - a pointer to the base of a new style expansion board.
    configDev - the ConfigDev structure that will be read in.
    offset - a logical offset from the configdev base
```

```
RESULTS
    error - If the board address does not contain a valid new style
            expansion board, then error will be non-zero.

EXAMPLES
    configDev = AllocConfigDev();
    if( ! configDev ) panic();

    error = ReadExpansionBoard( board, configDev );
    if( ! error ) {
            configDev->cd_BoardAddr = board;
            ConfigBoard( configDev );
    }

EXCEPTIONS
    Not typically called by user code.

SEE ALSO
    ReadExpansionByte, WriteExpansionByte

BUGS
```

1.18 expansion.library/ReleaseConfigBinding

```
NAME
    ReleaseConfigBinding - allow others to bind to drivers

SYNOPSIS
    ReleaseConfigBinding()

FUNCTION
    This call should be used when you are done binding drivers
    to ConfigDev entries.  It releases the SignalSemaphore; this
    allows others to bind their drivers to ConfigDev structures.

INPUTS

RESULTS

EXAMPLES

EXCEPTIONS

SEE ALSO
    ObtainConfigBinding

BUGS
```

1.19 expansion.library/RemConfigDev

```
NAME
    RemConfigDev - remove a ConfigDev structure from the system

SYNOPSIS
    RemConfigDev( configDev )
                  A0

FUNCTION
    This routine removes the specified ConfigDev structure from the
    list of Configuration Devices in the system.

INPUTS
    configDev - a valid ConfigDev structure.

RESULTS

EXCEPTIONS

SEE ALSO
    AddConfigDev

BUGS
```

1.20 expansion.library/SetCurrentBinding

```
NAME
    SetCurrentBinding - sets static board configuration area

SYNOPSIS
    SetCurrentBinding( currentBinding, size )
                       A0              D0:16

FUNCTION
    This function records the contents of the "currentBinding"
    structure in a private place.  It may be read via
    GetCurrentBinding().  This is really a kludge, but it is
    the only way to pass extra arguments to a newly configured
    device.

    A CurrentBinding structure has the name of the currently
    loaded file, the product string that was associated with
    this driver, and a pointer to the head of a singly linked
    list of ConfigDev structures (linked through the cd_NextCD
    field).

    Many devices may not need this information; they have hard
    coded into themselves their manufacture number.  It is
    recommended that you at least check that you can deal with
    the product code in the linked ConfigDev structures.

INPUTS
    currentBinding - a pointer to a CurrentBinding structure

    size - the size of the user's binddriver structure.  No
```

```
        more than this much data will be copied.  If size is
        larger than the libraries idea a CurrentBinding size,
        then the structure will be null padded.

RESULTS

EXAMPLES

EXCEPTIONS

SEE ALSO
    GetCurrentBinding

BUGS
```

1.21 expansion.library/WriteExpansionByte

```
NAME
    WriteExpansionByte - write a byte nybble by nybble.

SYNOPSIS
    error = WriteExpansionByte( board, offset, byte )
    D0                          A0     D0      D1

FUNCTION
    WriteExpansionByte write a byte to a new-style expansion
    board.  These boards have their writeable data organized
    as a series of nybbles in memory.  This routine writes
    two nybbles in a very carefull manner to work with all
    types of new expansion boards.

    To make certain types of board less expensive, an expansion
    board's write registers may be organized as either a
    byte-wide or nybble-wide register.  If it is nybble-wide
    then it must latch the less significant nybble until the
    more significant nybble is written.  This allows the
    following algorithm to work with either type of board:

            write the low order nybble to bits D15-D12 of
                    byte (offset*4)+2

            write the entire byte to bits D15-D8 of
                    byte (offset*4)

    The offset is a byte offset into a ExpansionRom structure.
    The actual memory address read will be four times larger.
    The macros EROFFSET and ECOFFSET are provided to help get
    these offsets from C.

INPUTS
    board - a pointer to the base of a new style expansion board.
    offset - a logical offset from the configdev base
    byte - the byte of data to be written to the expansion board.

RESULTS
```

```
    error - the routine will return a zero on success, non-zero if
            there was a problem.

EXAMPLES
    err = WriteExpansionByte( cd->BoardAddr, ECOFFSET( ec_Shutup ),  0 );
    err = WriteExpansionByte( cd->BoardAddr, ECOFFSET( ec_Interrupt ), 1 );

EXCEPTIONS
    Not typically called by user code.

SEE ALSO
    ReadExpansionByte, ReadExpansionRom

BUGS
```