

Fundamentals of FOSSIL implementation and use
Version 5, February 11, 1988
Rick Moore, Solar Wind Computing

FidoNet Address: Zone 1, Network 115, Node 333
(1:115/333)

Copyright (C) 1987, VEP Software, Naugatuck, CT 06770. All rights reserved.

Copyright (C) 1988, Rick Moore, Homewood, IL, 60430. All rights reserved.

This document may be freely used or copied by anyone interested in the data contained herein. No fees may be charged for distribution of this document. You will be held accountable for all such charges, and expected to either reimburse those persons or organizations so charged, or to make a donation in the exact amount of those fees to the International FidoNet Association, to assist them in their efforts to advance the technology of personal computer telecommunications.

Fundamentals of FOSSIL implementation and use

Page 2

Introduction

A. Objectives of this document

This document is directed at implementers or intellectuals. It is meant for use in implementing applications that can use FOSSIL drivers, or for details needed to implement a new FOSSIL. As such it won't always go out of its way to explain itself to the neophyte.

This document will have served its purpose to you if you are able to use the data contained within to perform either of the above tasks. If you feel that necessary data has been omitted please contact Rick Moore at the above listed address so that the appropriate changes can be made. Any lines changed in the current version are marked with "|" in the left margin.

B. Historical perspective

For those people who were not lucky enough to have an IBM PC or a system nearly completely compatible, the world has not been very friendly. With his implementation of the Generic Fido(tm) driver, Tom Jennings made it possible for systems that had nothing in common with an IBM PC except an 808x-class processor, and the ability to run MS-DOS Version 2 and above, to run his Fido(tm) software. That was a lot to ask, and a lot of people thought it was enough.

But not everyone. While Thom Henderson was debugging Version 4.0 of his SEAdog(tm) mail package, an "extended" Generic driver was designed (in cooperation with Bob Hartman) as a quick kludge to help him get past a problem with certain UART chips. The new hook was quickly pounced upon by Vince Perriello, who, with almost DAILY prodding (ouch! it still hurts) by Ken Kaplan, had been working with Henderson to get DEC Rainbow support into SEAdog. Vince then coded a driver to use this hook and - Voila! -

SEAdog 4.0 started working like a champ on the Rainbow.

At the same time something was rotten in the state of Texas. Wynn Wagner started encountering some serious difficulties in his Opus development effort. Specifically, he couldn't force the Greenleaf(tm) Communications Libraries to behave in exactly the way he felt Opus required. Enter Bob Hartman. Having already enjoyed success in the effort with Thom Henderson, he suggested to Wynn that with very few extensions, any driver that was already SEAdog(tm) 4.0 compatible could drive Opus as well. About that time, Vince called Wynn to discuss porting Opus to the DEC Rainbow. Wynn called Bob, Bob called Vince, and the FOSSIL driver came into existence.

FOSSIL is an acronym for "Fido/Opus/SEAdog Standard Interface Layer". To say that the concept has gained wide acceptance in the FidoNet community would be an understatement. Henk Wevers' DUTCHIE package uses the FOSSIL communications services. Ron Bemis' OUTER package uses FOSSIL services for everything it does and as a result it is completely generic. There are already FOSSIL implementations for the Tandy 2000, Heath/Zenith 100, Sanyo 555 and other "non-IBM" architectures. With each new 'port' of the spec, the potential of a properly coded FOSSIL application grows!

Fundamentals of FOSSIL implementation and use

Page 3

Basic conventions and calling method

C. Basic principles of a FOSSIL driver

1) Interrupt 14h.

The one basic rule that the driver depends upon, is the ability for ANY target machine to allow the vector for INT 14h (usually pointing to BIOS comm functions) to be "stolen" by the driver. In a system where the INT 14h vector is used already, it must be possible to replace the "builtin" functionality with that of a FOSSIL, when an application that wants the use of a FOSSIL is to be run on the target machine.

2) How to install a FOSSIL driver in a system

There's no hard and fast way to do this. The FOSSIL might be implemented as part of a device driver (like Ray Gwinn's X00.SYS) and therefore gets loaded using a line in CONFIG.SYS at bootup time. It might be done as a TSR (terminate and stay resident) program, in which event you install it by running the program (DECcomm by Vince Perriello and Opus!Comm by Bob Hartman work this way, for example).

3) How an application can detect the presence of a FOSSIL

The driver has a "signature" that can be used to determine whether it is present in memory. At offset 6 in the INT 14h service routine is a word, 1954h, followed by a byte that specifies the maximum function number supported by the driver. This is to make it possible to determine when a driver is present and what level of functionality it provides. Also, the Init call (see below) returns a 1954h in AX. SEAdog(tm) looks at the signature and Opus just goes for the Init. Fido doesn't do either.

4) How to call a FOSSIL function

The FOSSIL driver is entered by issuing a software Interrupt 14h from the application program. The code corresponding to the desired function should be in 8-bit register AH. For calls that relate to communications, the port number will be passed from the application in register DX. When DX contains a zero (0) it signifies use of COM1, or whatever the "first" serial port on your machine is called. A one (1) in DX points the driver at COM2, and so on. A value of 00FFh in DX is considered a special case where the driver should do no actual processing but return SUCCESS. In the specific case of Init/Uninit with DX=00FFh, the FOSSIL should perform all non-communications processing necessary with such calls. In some machines (H/Z-100 for example), the FOSSIL must assume control of the keyboard in order to service the keyboard functions.

**FOR ALL FUNCTIONS, ALL REGISTERS NOT SPECIFICALLY CONTAINING
A FUNCTION RETURN VALUE MUST BE PRESERVED ACROSS THE CALL.**

Fundamentals of FOSSIL implementation and use

Page 4

Communications functions

D. Functions currently defined for FOSSILs

AH = 00h Set baud rate

Parameters:

Entry: AL = Baud rate code

DX = Port number

Exit: AX = Port status (see function 03h)

This works the same as the equivalent IBM PC BIOS call, except that it ONLY selects a baud rate. This is passed in the high order 3 bits of AL as follows:

010	=	300	baud
011	=	600	' '
100	=	1200	' '
101	=	2400	' '
110	=	4800	' '
111	=	9600	' '
000	=	19200	' ' (Replaces old 110 baud mask)
001	=	38400	' ' (Replaces old 150 baud mask)

The low order 5 bits can be implemented or not by the FOSSIL, but in all cases, if the low order bits of AL are 00011, the result should be that the communications device should be set to eight data bits, one stop bit and no parity. This setting is a MINIMUM REQUIREMENT of Fido, Opus and SEAdog. For purposes of completeness, here are the IBM PC "compatible" bit settings:

Bits 4-3 define parity:	0 0	no parity
	1 0	no parity
	0 1	odd parity
	1 1	even parity

Bit 2 defines stop bits:	0	1 stop bit;
	1	1.5 bits for 5-bit char; 2 for others

Bits 1-0 character length:	0 0	5 bits
----------------------------	-----	--------

0 1	6 bits
1 0	7 bits
1 1	8 bits

Fundamentals of FOSSIL implementation and use
Page 5
Communications functions

AH = 01h Transmit character with wait

Parameters:

Entry: AL = Character
 DX = Port number

Exit: AX = Port status (see function 03h)

AL contains the character to be sent. If there is room in the transmit buffer the return will be immediate, otherwise it will wait until there is room to store the character in the transmit buffer. On return, AX is set as in a status request (see function 03h).

AH = 02h Receive character with wait

Parameters:

Entry: DX = Port number

Exit: AH = 00h
 AL = Input character

If there is a character available in the receive buffer, returns with the next character in AL. It will wait until a character is received if none is available.

AH = 03h Request status

Parameters:

Entry: DX = Port number

Exit: AX = Status bit mask (see below)

Returns with the line and modem status in AX. Status bits returned are:

```

    In AH:
        Bit 0 = RDA  - input data is available in buffer
        Bit 1 = OVRN - the input buffer has been overrun.
|
All
|
|           characters received after the
buffer is
|
|           full should be discarded.
        Bit 5 = THRE - room is available in output buffer
        Bit 6 = TSRE - output buffer is empty

    In AL:
|
|           Bit 3 = Always 1 (always return with this bit set
to 1)
|
|           Bit 7 = DCD  - carrier detect
```

This can be used by the application to determine whether carrier detect (CD) is set, signifying the presence/absence of a remote connection, as well as monitoring both the input and output buffer status. Bit 3 of AL is always returned set to enable programs to use it as a carrier detect bit on hardwired (null modem) links.

Fundamentals of FOSSIL implementation and use

Page 6

Communications functions

AH = 04h Initialize driver

Parameters:

Entry: DX = port number

(BX = 4F50h

ES: CX = ^C flag address --- optional)

Exit: AX = 1954h if successful

BL = maximum function number supported
(not counting functions 7Eh and

above)

BH = rev of FOSSIL doc supported

This is used to tell the driver to begin operations, and to check that the driver is installed. This function should be called before any other communications calls are made. At this point all interrupts involved in supporting the comm port (specified in DX) should be set up for handling by the FOSSIL, then enabled. If BX contains 4F50h, then the address specified in ES: CX is that of a ^C flag byte in the application program, to be incremented when ^C is detected in the keyboard service routines. This is an optional service and only need be supported on machines where the keyboard service can't (or won't) perform an INT 1Bh or INT 23h when a Control-C is entered. DTR is raised by this call. The baud rate must NOT be changed by this call.

NOTE: Should an additional call to this service occur (2 Inits or Init, Read, Init, etc.) the driver should reset all buffers, flow control, etc. to the INIT state and return SUCCESS.

AH = 05h Deinitialize driver

Parameters:

Entry: DX = Port number

Exit: None

This is used to tell the driver that comm port operations are ended. The function should be called when no more comm port functions will be used on the port specified in DX. DTR is NOT affected by this call.

AH = 06h Raise/lower DTR

Parameters:

Entry: DX = Port number

AL = DTR state to be set (01h = Raise, 00h =
Lower)

Exit: None

This function is used to control the DTR line to the modem. AL = 00h means lower DTR (disable the modem), and AL = 01h means to raise DTR (enable the modem). No other function (except Init) should alter DTR.

Fundamentals of FOSSIL implementation and use

Page 7

Communications functions

AH = 07h Return timer tick parameters

Parameters:

Entry: None

Exit: AL = Timer tick interrupt number

 AH = Ticks per second on interrupt number
 in AL

 DX = Approximate number of milliseconds
 per tick

This is used to determine the parameters of the timer tick on any given machine. Three numbers are returned:

AL = Timer tick interrupt number

AH = Ticks per second on interrupt number shown in AL

DX = Milliseconds per tick (approximate)

Applications can use this for critical timing (granularity of less than one second) or to set up code (such as a watchdog) that is executed on every timer tick. See function 16h (add/delete function from timer tick) for the preferred way of actually installing such code.

AH = 08h Flush output buffer

Parameters:

Entry: DX = Port number

Exit: None

This is used to force any pending output. It does not return until all pending output has been sent. You should use this call with care. Flow control (documented below) can make your system hang on this call in a tight an interruptible loop under the right circumstances.

AH = 09h Purge output buffer

Parameters:

Entry: DX = Port number

Exit: None

This is used to purge any pending output. Any output data remaining in the output buffer (not transmitted yet) is discarded.

AH = 0Ah Purge input buffer

Parameters:

Entry: DX = Port number

Exit: None

This is used to purge any pending input. Any input data which is still in the buffer is discarded.

Fundamentals of FOSSIL implementation and use

Page 8

Communications functions

AH = 0Bh Transmit no wait

Parameters:

Entry: DX = Port number

Exit: AX = 0001h - Character was accepted
 = 0000h - Character was not accepted

This is exactly the same as the "regular" transmit call, except that if the driver is unable to buffer the character (the buffer is full), a value of 0000h is returned in AX. If the driver accepts the character (room is available), 0001h is returned in AX.

AH = 0Ch Non-destructive read-ahead

Parameters:

Entry: DX = Port number

Exit: AH = 00h - Character is
 AL = Next character available
 AX = FFFFh - Character is not
 available

Return in AL the next character in the receive buffer. If the receive buffer is empty, return FFFFh. The character returned remains in the receive buffer. Some applications call this "peek".

AH = 0Dh Keyboard read without wait

Parameters:

Entry: None

Exit: AX = IBM-style scan code (Character
 available)
 = FFFFh (Character not
 available)

Return in AX the next character (non-destructive read ahead) from the keyboard; if nothing is currently in the keyboard buffer, return FFFFh in AX. Use IBM-style function key mapping in the high order byte. Scan codes for non-"function" keys are not specifically required, but may be included. Function keys return 00h in AL and the "scan code" in AH.

AH = 0Eh Keyboard read with wait

Parameters:

Entry: None

Exit: AX = IBM-style scan code

Return in AX the next character from the keyboard; wait if no character is available. Keyboard mapping should be the same as function 0Dh.

Fundamentals of FOSSIL implementation and use

Page 9

Communications functions

AH = 0Fh Enable or disable flow control

Parameters:

Entry: AL = Bit mask describing requested flow control

DX = Port number

Exit: None

TRANSMIT flow control allows the "other end" to restrain the transmitter when you are over-running it. RECEIVE flow control tells the FOSSIL to attempt to do just that if it is being overwhelmed.

Two kinds of basic flow control are supported:

Bit 0 = 1	Xon/Xoff on transmit
Bit 1 = 1	CTS/RTS (CTS on transmit, RTS on receive)
Bit 2	Reserved
Bit 3 = 1	Xon/Xoff on Receive

Flow control is enabled, or disabled, by setting the appropriate bits in AL for the types of flow control we want to ENABLE (value = 1), and/or DISABLE (value = 0), and calling this function. Bit 2 is reserved for DSR/DTR, but is not currently supported in any implementation.

Enabling transmit Xon/Xoff will cause the FOSSIL to stop transmitting upon receiving an Xoff. The FOSSIL will resume transmitting when an Xon is received.

Enabling CTS/RTS will cause the FOSSIL to cease transmitting when CTS is lowered. Transmission will resume when CTS is raised. The FOSSIL will drop RTS when the receive buffer reaches a predetermined percentage full. The FOSSIL will raise RTS when the receive buffer empties below the predetermined percentage full. The point(s) at which this occurs is left to the individual FOSSIL implementor.

Enabling receive Xon/Xoff will cause the FOSSIL to send a Xoff when the receive buffer reaches a pre-determined percentage full. An Xon will be sent when the receive buffer empties below the pre-determined percentage full. The point(s) at which this occurs is left to the individual FOSSIL implementor.

Applications using this function should set all bits ON in the high nibble of AL as well. There is a compatible (but not identical) FOSSIL driver implementation that uses the high nibble as a control mask. If your application sets the high nibble to all ones, it will always work, regardless of the method used by any given driver.

Fundamentals of FOSSIL implementation and use

Page 10

Communications functions

AH = 10h Extended Control-C / Control-K checking and transmit on/off

Parameters:

Entry: AL = Bit mask (see below)

DX = Port number

Exit: AX = 0001h - Control-C/K has been received
 = 0000h - Control-C/K has not been received

This is used for BBS operation, primarily. A bit mask is passed in AL with the following flags:

Bit 0 Enable/disable Control-C / Control-K checking

Bit 1 Disable/enable the transmitter

The Enable (bit 0 = 1) and Disable (Bit 0 = 0) Control C/Control-K check function is meant primarily for BBS use. When the checking is enabled, a Control-C or Control-K received from the communications port will set a flag internal to the FOSSIL driver, but will not be stored in the input buffer. The next use of this function will return the value of this flag in register AX then clear the flag for the next occurrence. The returned value is used by the BBS software to determine whether output should be halted or not.

The Disable (Bit 1 = 1) and Enable (Bit 1 = 0) Transmitter function lets the application restrain the asynchronous driver from output in much the same way as XON/XOFF would.

AH = 11h Set current cursor location.

Parameters:

Entry: DH = Row (line)

DL = Column

Exit: None

This function looks exactly like like INT 10h, subfunction 2, on the IBM PC. The cursor location is passed in DX: row in DH and column in DL. The function treats the screen as a coordinate system whose origin (0,0) is the upper left hand corner of the screen.

AH = 12h Read current cursor location.

Parameters:

Entry: None

Exit: DH = Row (line)

DL = Column

Looks exactly like INT 10h, subfunction 3, on the IBM PC. The current cursor location (using the same coordinate system as function 16h) is passed back in DX.

Fundamentals of FOSSIL implementation and use

Page 11

Communications functions

AH = 13h Single character ANSI write to screen.

Parameters:

Entry: AL = Character to display

Exit: None

The character in AL is sent to the screen by the fastest method possible that allows ANSI processing to occur (if available). This routine should not be used in such a way that DOS output (which is not re-entrant) can not be employed by some FOSSIL driver to perform the function (in fact, on the IBM PC that is likely to be how it's done). On some systems such as the DEC Rainbow this will be a very fast method of screen writing.

AH = 14h Enable or disable watchdog processing

Parameters:

Entry: AL = 01h - Enable watchdog

 = 00h - Disable watchdog

 DX = Port number

Exit: None

When watchdog is enabled, the state of the carrier detect (CD) line on the comm port specified in DX should be constantly monitored. Should the state of that line become FALSE (carrier lost), the system should be re-booted, to enable the BBS (or other application) to start up again. This monitor is not affected by Init/Uninit etc.

AH = 15h Write character to screen using BIOS support routines

Parameters:

Entry: AL = Character to display

Exit: None

The character in AL is sent to the screen using BIOS-level Input/Output routines. This differs from function 13h in that DOS I/O CAN NOT be used, as this function might be called from driver level.

Fundamentals of FOSSIL implementation and use

Page 12

Communications functions

AH = 16h Insert or delete a function from the timer tick chain

Parameter:

Entry: AL = 01h - Add a function
 = 00h - Delete a function
 ES = Segment of function
 DX = Offset of function
Exit: AX = 0000h - Operation successful
 = FFFFh - Operation unsuccessful

This function is used to allow a central authority to manage the timer interrupts, so that as code is loaded and unloaded, the integrity of the "chain" is not compromised. Rather than using the traditional method of saving the old contents of the timer vector, storing the address of your routine there, and executing a far call to the "old" routine when yours is done, instead you call this function. It manages a list of such entry points and calls them on a timer tick (interrupt) using a FAR call. All the usual cautions about making DOS calls apply (that is, DON'T!).

This makes it possible for a program to get in and out of the tick chain without having to know whether another program has also done so since it first insinuated itself. At least 4 entries should be available in the driver's table (including one to be used by Watchdog if implemented that way).

AH = 17h Reboot system

Parameters:

Entry: AL = 00h - "Cold boot"
 = 01h - "Warm boot"

Perform the old 3-finger salute. Used in extreme emergency by code that can't seem to find a "clean" way out of the trouble it has gotten itself into. Hopefully it won't happen while you're computing something in the other half of a DoubleDOS system. If your machine can make a distinction

between a "cold" (power-up, self-test and boot) and a "warm" (just boot) bootstrap, your FOSSIL should support the flag in AL. Otherwise just do whatever bootstrap is possible.

Fundamentals of FOSSIL implementation and use
Page 13
Communications functions

| AH = 18h Read block (transfer from FOSSIL to user buffer)

| Parameters:

| Entry: CX = Maximum number of characters to
transfer

| DX = Port number

| ES = Segment of user buffer

| DI = Offset into ES of user buffer

| Exit: AX = Number of characters actually
transferred

A "no-wait" block read of 0 to FFFFh characters from the FOSSIL inbound ring buffer to the calling routine's buffer. ES:DI are left unchanged by the call; the count of bytes actually transferred will be returned in AX.

| AH = 19h Write block (transfer from user buffer to FOSSIL)

| Parameters:

| Entry: CX = Maximum number of characters to
transfer

| DX = Port number

| ES = Segment of user buffer

| DI = Offset into ES of user buffer

| Exit: AX = Number of characters actually
transferred

A "no-wait" block move of 0 to FFFFh characters from the calling program's buffer into the FOSSIL outbound ring buffer. ES:DI are left unchanged by the call; the count of bytes actually transferred will be returned in AX.

| AH = 1Ah Break begin or end

| Parameters:

| Entry: AL = 01h - Start sending 'break'

| = 00h - Stop sending 'break'

| DX = port number

Exit: None

Send a break signal to the modem. If AL=01h the driver will commence then transmission of a break. If AL=00h the driver will end the break. This is useful for communications with devices that can only go into 'command mode' when a BREAK is received. Note: the application is responsible for the timing of the BREAK. Also, if the FOSSIL has been restrained by an Xoff received from the modem, the flag will be cleared. An Init or Un-Init will stop an in-progress BREAK.

Fundamentals of FOSSIL implementation and use

Page 14

Communications functions

| AH = 1Bh Return information about the driver

| Parameters:

| Entry: CX = Size of user info buffer in bytes

| DX = Port number

| ES = Segment of user info buffer

| DI = Offset into ES of user info buffer

| Exit: AX = Number of bytes actually transferred

Transfer information about the driver and its current status to the user for use in determining, at the application level, limits of the driver. Designed to assist "generic" applications to adjust to "foreign" gear.

| The data structure currently returned by the driver is as follows (sorry

| but you'll have to live with assembly syntax):

```
|        info        equ        $                        ; define begin of
structure
|        strsiz    dw        info_size                ; size of the structure in
bytes
|        majver    db        curr_fossil             ; FOSSIL spec driver
conforms to
|        minver    db        curr_rev                ; rev level of this
specific driver
|        ident     dd        id_string               ; "FAR" pointer to ASCII
ID string
|        ibufr     dw        ibsize                  ; size of the input buffer
(bytes)
|        ifree     dw        ?                        ; number of bytes left in
buffer
|        obufr     dw        obsize                  ; size of the output
buffer (bytes)
|        ofree     dw        ?                        ; number of bytes left in
the buffer
|        swidth    db        screen_width           ; width of screen on this
adapter
|        sheight   db        screen_height          ; height of screen        "
```

```
|      baud      db      ?      ; ACTUAL baud rate,  
computer to modem  
|      info_size equ $-info
```

The ident string should be null-terminated, and NOT contain a newline. The baud rate byte contains the bits that Function 00h would use to set the port to that speed.

The fields related to a particular port (buffer size, space left in the buffer, baud rate) will be undefined if port FFh or an invalid port is contained in DX.

Additional information will always be passed after these, so that, for example, offset "sheight" will never change with FOSSIL revision changes.

Fundamentals of FOSSIL implementation and use

Page 15

"Layered Application" services

The functions below are not necessarily FOSSIL related. However, because dispatchers that support them are hooked on Interrupt 14H, it behooves the FOSSIL developer to support them as well to avoid fragmenting memory with several dispatchers.

```
| AH = 7Eh      Install an "external application" function
|
|      Parameters:
|      Entry:   AL = Code assigned to external application
|               DX = Offset of application entry point
|               ES = Segment of application entry point
|      Exit:    AX = 1954h
|               BL = Code assigned to application (same as
input AL)
|               BH = 01h - Installation was successful
|               = 00h - Installation failed
```

This call is used by external application code (special screen drivers, modem code, database code, etc) to link into the INT 14h service for use by multiple applications. The "error return" (BH=0 with AX=1954h) should mean that another application layer has already been installed at that particular code. Codes 80h through BFh should be supported.

External application codes 80h-83h are reserved by FOSSIL developers for re-organizing FOSSIL services by type (comm, screen, keyboard, system).

Installed application code will be entered, via a FAR call, from the INT 14H dispatcher whenever it is entered with H=(application code).

If the value returned in AX from this function is not 1954h, the service code that is trying to be installed should bring up its own INT 14h code that can service INT 14h functions 7h-BFh (80h-BFh are "applications").

```

|   AH = 7Fh      Remove an "external application" function
|
|       Parameters:
|           Entry:  AL = Code assigned to external application
|                   DX = Offset of application entry point
|                   ES = Segment of application entry point
|           Exit:   AX = 1954h
|                   BL = Code assigned to application (same as
input AL)
|                   BH = 01h - Removal was successful
|                   = 00h - Removal failed
|

```

Removes an application's entry into the table. Usually so it can remove itself from memory. Error return means ES:DX did not match or that there is no entry at the slot described by AL.

An application that wants to remove itself from memory can issue the 7F function to remove itself from the table, then, if it is successful, get out of memory. If it had to install itself with an INT 14h dispatcher it may back itself out, provided no other applications have been installed on top of it (using its dispatcher).

Fundamentals of FOSSIL implementation and use
Page 16

E. Validation Suite.

Well, there is one, but it's involved. Here is a list of software that is known to use FOSSIL calls, and the range of calls used by that software:

Software package	Fossil calls used
Fido, V11w, generic version	00h - 07h
SEAdog, V4.1b	00h - 0Eh
Opus, V1.03a	00h - 17h
BinkleyTerm, V1.30	00h - 1Bh

While there is certainly no guarantee that your FOSSIL is bug-free if all the above software runs with it, you have probably done as much as you can in a test environment if your FOSSIL is tested with each of these packages.

F. Technical Discussion.

A FOSSIL echomail conference exists, for the purpose of exchanging info and implementation details for FOSSIL drivers. It is coordinated by Ray Gwinn at FidoNet node 1:109/639. Contact him for details on how to join. Keep in mind though, that this conference is intended SPECIFICALLY for implementors of FOSSIL software and not as a general Q&A conference for people who think FOSSILs have something to do with paleontology.

G. Distribution Of This Document.

This document may be distribute freely as long as it is not modified in any way. Please list all changes and deviations in a given FOSSILimplementation in an addendum contained in a separate file added to the

FOSSIL archive. Also, please do not distribute this document without the accompanying version of FOSSIL.CHT. This will help avoid confusion, among both FOSSIL implementors and application developers.