New Technical Notes

Macintosh

Developer Support

Serial Driver Q&As

Devices

Revised by: Developer Support Center June 1993
Written by: Developer Support Center October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

New Q&As in this Technical Note:
Replacing Macintosh serial drivers

Replacing Macintosh serial drivers
Date written: 11/4/92
Last reviewed: 4/1/93

I have some problems writing a driver/INIT/cdev combination. I want to load a driver that is of the same type as the serial drivers into memory. What's the correct sequence for loading a driver into memory without opening it? What are the calls a serial driver has to reply to? Because my driver needs common stuff, I wanted to use *one* driver instead of two (as .AIn and .AOut). Is it OK to register the same driver within the Communications Toolbox? If it's not OK, would it be OK to write a small output driver that just calls the main driver (which is registered as input driver) while writing? Or would it be better to use three drivers, one for output, one for input, and one that does all the stuff?

—

From your INIT, you'll want to load the driver's resource into the system heap and detach it with DetachResource; it can then install it into the unit table by finding an empty entry and putting it there with the DrvrInstall call, as described in the Macintosh Technical Note

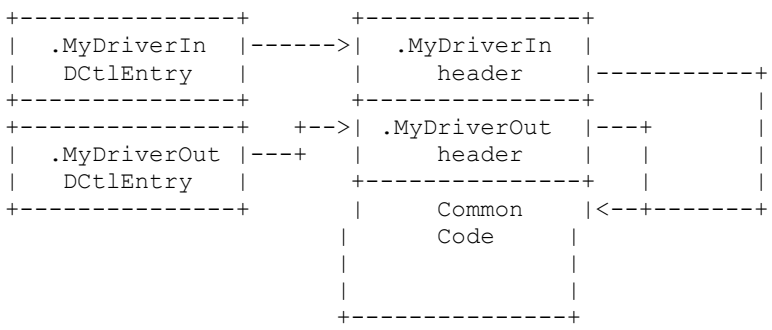
“_AddDrive, _DrvrInstall, and _DrvrRemove.”)

A serial driver should respond to all the calls supported by the standard serial driver, as described in the chapters on the Serial Driver in *Inside Macintosh* Volumes II and IV, along

with a couple of new features described in the “System 7.0 serial driver changes” in the Macintosh “Serial Driver Q&As” Tech Note.

You should really use two drivers for your serial driver; this is important because the Device Manager maintains only one active transaction for each driver at any time; thus, if you tried to share one driver between the input and the output, it would be half-duplex; it couldn’t send and receive simultaneously. The best way for your drivers to communicate would be for them to share a data block; have one hold the data itself, and the other have a pointer to the data. If you want to share common code, the best way to do it is to share the driver code. Each driver must have its own driver handle—the drivers need to have unique names, because they’re opened by name. Because the driver’s name is embedded in its code, each code handle must be separate from the other.

Basically, here’s what you should do: When loading your drivers, make them ROM-based; set bit 6 in the flags field of the driver’s DCtlEntry. The only thing this means is that the dCtlDriver field is a pointer to the driver, not a handle. You can then load all your drivers in one block that shares code. The block would look like this (excuse the ugly ASCII graphics):



As you can see, each of the two different DCtlEntries points to a different driver header within the same block, each of which points to the common code that also resides in that block. In this way, you can easily share code while satisfying the need for a separate driver header for each driver.

A similar problem occurs when registering drivers under different names, because once again the name is embedded within the driver; you can use the same solution as above, only you’ll have many more than two headers sharing the same code.

Serial driver -90 error code

Date written: 12/23/92
Last reviewed: 3/1/93

Sometimes, at the beginning of a PBRead on a serial port, I get back a result code of -90 in the completion routine. I don’t quite know how to handle this error, because I can’t find a -90 result code anywhere. Any idea?

—

According to the MPW Errors.h interface file, -90 is a BreakRecd result. (The interface files are always a good place to look for error codes and calls that you can’t find.) The serial driver returns that error to a pending Read if the SCC chip detects a break condition.

Serial driver bug keeps DTR from negating across CloseDriver call
Date written: 11/19/92
Last reviewed: 3/1/93

We've noticed that the RS-232 DTR lead can be *asserted* when CloseDriver is called to shut down the serial communications transmission driver. We'd like to be able to keep the state of DTR unchanged across a call to close the driver. According to *Inside Macintosh* Volume IV, page 226, the advanced serial driver control call "will cause DTR to remain unchanged when the driver is closed." We've been successful at keeping DTR asserted across a call to CloseDriver using this call but haven't been able to keep DTR negated. We use the following call sequence:

1. Call OpenDriver to open the serial output driver. DTR is asserted.
2. Make an advanced serial driver control call (csCode = 16) via a Device Manager PBControl call. We set the first byte of the csParam array to 0x80.
3. Make an advanced serial driver control call (csCode = 18) to Negate DTR. DTR is negated.
4. Call CloseDriver to shut down the serial output driver. DTR is asserted.

Is there a way to keep DTR negated across the CloseDriver call?

—

You found a bug—a bug that's been around for at least six years. Probably no one else found it because they always want DTR to be asserted on close.

The problem is that the serial driver maintains two globals, but when you call the Negate DTR control call, it updates only one of them. The one it fails to update is used to reinitialize the SCC on driver close. And that's why the DTR signal gets asserted.

As a workaround until the bug is fixed, if you know you don't want DTR asserted on close, don't call Control 16. Or, call it, but before closing the driver, call it again, clearing bit 7 (to "undo" leaving DTR unchanged).

Use serShk instead of SerHShake for serial hardware handshaking
Date written: 9/25/92
Last reviewed: 11/1/92

We're trying to connect an external device to a Macintosh via the modem serial port. We can receive data from the Macintosh but are having trouble with data flow handshake. We have asserted the input handshake line (CTS), but data continues to be sent from the Macintosh.

—

You probably are attempting to use the SerHShake control call (csCode = 10) to turn hardware handshaking on, as described in *Inside Macintosh* Volume II. Instead, you should be using the serShk call, as described in *Inside Macintosh* Volume IV, page 226, which will allow you to turn on DTR handshaking, which will allow you to use flow control to limit the output of the Macintosh.

Close Macintosh serial driver after OpenDriver call

Date Written: 5/5/92
Last reviewed: 9/22/92

Inside Macintosh says that ROM drivers opened with OpenDriver shouldn't be closed. However, it seems that any driver opened with OpenDriver should be closed when the application is done. Should our application close the serial port after using it?

—

The *Inside Macintosh* statement was made in 1984 when the SCC serial chip generated mouse movement interrupts as part of the quadrature scheme. Closing the serial driver turned off those interrupts, bringing mouse movement to a squeaking halt. With the advent of the Apple Desktop Bus (ADB), the SCC and the serial driver no longer have anything to do with mouse movement. For this reason, the *Inside Macintosh* statement should be revised to say, "If you're running on a Macintosh Plus or earlier model, it's recommended you don't close the ROM serial driver; otherwise please do close it." Note that the (now obsolete) RAMSDOpen and RAMSDClose routines did the right thing to preserve the mouse interrupts: RAMSDOpen closed the ROM driver but immediately opened the RAM driver (which was an updated ROM driver), and RAMSDClose closed the RAM driver and reopened the ROM driver.

As a general rule, applications that open the serial driver with OpenDriver should do so only when they're actually going to use it, and they should close it when they're done. (Note that it's important to do a KillIO on all I/O before closing the serial port!) There are a couple of reasons for closing the port when you're finished using it. First, it conserves power on the Macintosh portable models; while the serial port is open the SCC drains the battery. Second, closing the serial port avoids conflicts with other applications that use it. *Inside Macintosh* is incorrect in stating that you shouldn't close the port after issuing an OpenDriver call.

Most network drivers shouldn't be closed when an application quits, on the other hand, since other applications may still be accessing the driver.

New Macintosh serial driver status and control calls

Date Written: 4/28/92
Last reviewed: 7/13/92

Are there any new control or status calls available in the Macintosh serial driver under System 7.0?

—

In addition to the *Inside Macintosh* serial driver documentation, support was added in System 7.0. The changes are implemented as patches to _Control and _Status. Here is further documentation:

1. Two status calls, opcodes $0009 and $8000, have been added. They both return the version number for the driver. The version number is returned as the first byte in the csParam field. There is no upper-level interface for this new call, only an assembly language interface.

2. The miscellaneous Control call (decimal 16 or $10) has been modified to add support for external clocking. The SCC chip has a mode in which an external device can drive its clock (as opposed to an internal clock) thru the CTS handshake line (HSKi). Bit 6 of the csParam now

controls the switch between internal and external clocking (bit 6 used to be reserved). This was needed for the Personal LaserWriter LS. The bit values are:

bit 6 = 0 means internal clocking. This is the default state.
bit 6 = 1 means external clocking.

As before, bit 7 controls the DTR line on Close of the driver. As before, bits 0–5 are reserved for future use.

3. The SerStatus Status call (8) has been modified to report a break condition. Bit 3 of the CumErrs byte has been changed from being reserved to being an indicator of whether a Break condition occurred. The bit values are:

bit 3 = 1 if a break was received.
bit 3 = 0 if no break was received.

Separate transmit & receive baud rates for Macintosh serial port
Date Written: 5/29/91
Last reviewed: 9/16/91

The Serial Driver does not support separate baud rates for transmit and receive. This is due to limitations imposed by the Z8530 SCC.

The SCC supports separate transmit and receive clocks; each is independently configurable to use a clock from the RTxC pin, the TRxC pin, the internal baud rate generator, or the output of the onboard DPLL. The first two options provide only a very limited, fixed set of baud rates. The last is used primarily for synchronous protocols. For maximum baud rate flexibility, the Serial Driver uses the internal baud rate generator for both transmit and receive. In some scenarios, separate transmit and receive rates would require two baud rate generators per channel, but there is only one.

Also, because the input and output drivers for a particular serial channel share code and data structures, commands like SerReset and SerHShake operate on both channels, regardless of the input or output designation corresponding to the refNum. In spite of the historical admonitions to make these calls separately for each driver, it’s not necessary to do so.

There are essentially only two options for running separate transmit/receive baud rates: operate in a half-duplex mode and SerReset between sending and receiving (gross and disgusting), or reprogram the SCC (risky) to use the RTxC clock or some other clocking option for either the transmitter or receiver, and live with one of the limited baud rate options that offers you. Baud rate options are limited because the standard input clock you have to depend on is 3.672 MHz. You could divide this down by factors of 1, 16, 32, or 64, giving you a minimum baud rate of 57.375 kbaud—not very helpful if you need something like 2400 or 9600 baud. You could run an external clock, but this adds expense and complexity.

Macintosh serial driver and synchronous communication

Date Written: 4/2/91
Last reviewed: 5/21/91

What Macintosh system call should I use to use the modem port for synchronous communication?

—

The Macintosh serial driver doesn't support synchronous communication. The only way to perform synchronous communication is to manipulate the hardware directly, which is unsupported. This means that if you directly manipulate the hardware, your product won't work on Macintosh systems that don't allow direct access to the Serial Communications Chip (SCC), such as the Macintosh IIfx.

Macintosh serial hardware handshaking

Date Written: 12/18/90
Last reviewed: 1/17/91

What is the method of asserting and then negating DTR with csCode = 17 and csCode = 18 respectively, in C on the Macintosh?

—

Open the serial output driver and get its refNum followed by:

```
ctrlErr = Control(refNum, csCode, &csParam);
```

The results depends on what csCode is used. For the csCodes 17 and 18, csParam is not used and could probably even be NULL.

As an example, it is necessary to use csCode = 14 instead of SerHShake() if you want to do full hardware handshaking. The code to do so is:

```
setHskErr = Control(gOutRefNum, kSerHShakeDTR, &hskFlags);
```

assuming that kSerHShakeDTR = 14 and hskFlags is a SerShk record.

Serial driver DCE dCtlFlags and serial port arbitration

Date Written: 12/3/90
Last reviewed: 2/6/91

How do I tell which serial ports are available? DTS's Macintosh Technical Note "Opening the Serial Driver" says not to check SPConfig, PortAUse, or PortBUse. If I do an OpenDriver on both ports to see which ones return no error, I may have two serial ports open, and I'm not supposed to close them. When I set the handshaking options on the output port, am I also setting them on the input port?

—

Serial port arbitration… such a lofty goal. The Communications Toolbox sort of does it, and you can do it in about the same way if you simply look at bit 5 of the low-order byte of the dCtlFlags field of the Serial Driver's DCE *(Inside Macintosh* Volume II, p. 190).

If the driver is already open, assume someone else is using it. When you use it, be sure to close it when you're done.

I'm not sure what you mean by you're "not supposed to close them" unless you're referring to the ancient serial driver documentation in *Inside Macintosh* Volume II. You can simply call CloseDriver these days as long as you're using a reasonably new serial driver (everything later than the 128K Macintosh Plus ROM).

If you want to be precise, you can call Status on the serial output driver with a csCode of 1 to get a handle to the driver's DCE. If the return value in csParam is a valid handle, you can dereference it and check the byte at offset dCtlQueue+1 (=7) to get the driver version number. If it's not zero, it's OK to close the driver.

That in itself should help you a lot.

It's not entirely obvious to what refNum one should address the SerHShake command, so you'll have to trust me. Just call it once on the output driver and everything will be OK. The serial driver doesn't support separate baud rates and handshaking protocols for each half of the duplex. (Ironically, I do recommend that you call SerReset on both halves. Don't ask me why.)

Where to get "Sample Serial NuBus Driver Code" disk
Date Written: 12/6/90
Last reviewed: 2/6/91

How can I get a copy of the Macintosh "Sample Serial NuBus Driver Code" disk mentioned on page vii of the *Apple Serial NuBus Card Programmer's Guide* (APDA # M0941LL/A)?

—

After doing some rather extensive checking I have been informed that the Driver Code is available through APDA. The product quoted is the Serial NB Tool, Product Number M0950LL/A. The tool is available for $40.00 and includes a disk and documentation.

Specifying a specific Macintosh serial baud rate
Date Written: 5/3/89
Last reviewed: 12/17/90

How can I program the Macintosh Serial Driver to use a baud rate of 38,400?

—

Use a baud rate constant of 1 for 38,400 baud. Note that you can specify a given baud rate by making the Serial Driver control call with a control code (csCode) of 13. The driver will return the closest baud rate that the Serial Driver will generate. Some trivia:

The formula for figuring out the SerConfig baud rate constants is

$$\text{Baud Rate Constant} = \frac{}{\text{Baud Rate}} - 2.0$$

For example, the baud rate constant for 9600 baud is

$$\text{Baud Rate Constant} = \frac{114709}{} - 2.0 = 11.949 - 2.0 = 9.949 = 10$$

```
                          9600
```

The 114709 figure comes from the SCC clock (3.670702 MHz) which comes from the system clock (15.6672 MHz), but doesn't divide down evenly because of the timing PALs (in fact, 3.670702 = 15667200 /4 * 15/16, where the 15/16 is due to PAL timing). For the Macintosh XL (Lisa), use 115200 instead of 114709.

Similarly, you can do a reverse calculation and find the baud rate, given a constant. Reversing the formula, you get

```
                        114709
Baud Rate = --------------------------
             Baud Rate Constant + 2.0
```

So, using the constant 10 (for 9600 baud), the formula gives

```
                        114709
Baud Rate = -------------------------=  9559.083
                       10 + 2.0
```

This isn't exactly 9600 baud. In fact, the error percentage is

```
           9559.083
1.0 -------------= 1.0 - 0.996 = 0.426%
             9600
```

If you try this with 38400, you get

```
                         114709
Baud Rate Constant = --------- 2.0 = 2.987 - 2.0 = 0.987 = 1,
                          38400
```

a baud rate of

```
                        114709
Baud Rate = -------------------------- = 38236.333,
             Baud Rate Constant + 2.0
```

and an error percentage of

```
           38236.333
1.0 -------------= 1.0 - 0.996 = 0.426%
             38400
```

X-Ref:
"The Serial Drivers: Advanced Control Calls," *Inside Macintosh,* Volume II

Opening the Serial Driver more than once and the PortInUse error

Date Written: 11/17/89
Last reviewed: 12/17/90

Shouldn't I get a PortInUse error if I open the Macintosh Serial Driver twice?

—

Arbitration is only done on *different* types of drivers. Say you try to open the Macintosh ROM asynchronous driver; it will fail *only* if the port is in use by a driver that is something other than async. If the port is in use by another asynchronous driver, you won't get an error.

For example, if you try to open the async ROM driver while AppleTalk (which is a different type of driver) is using it, you *will* get an error.

This unfortunately means users can open several different programs that use the serial port under MultiFinder, and, because they all use the Macintosh async driver, get no errors. Of course, all programs can then merrily change the SCC like crazy. That is why it has to be documented that users can't run multiple applications that use the Serial Driver under MultiFinder.

X-Ref:
Macintosh Technical Note "Opening the Serial Driver"