Apple II
Technical Notes

Developer Technical Support

# Apple IIGS
# #26: ROM Revision Summary

Revised by: Matt Deatherage September 1989
Written by: Rilla Reynolds October 1987

This Technical Note summarizes revisions to the Apple IIGS ROM.
**Changes since November 1988:** Revised to cover ROM 3.

---

Apple currently supports two configurations of the Apple IIGS ROM, ROM 1 and ROM 3. In August 1989, Apple IIGS computers began shipping with a 256K ROM, referred to as version 3 or ROM 3 (ROM 2 was skipped since there was already enough confusion about the first version, ROM 0, and the second version, ROM 1). System Software continues to support ROM 1, but it no longer supports ROM 0. Authorized Apple dealers can upgrade older systems (i.e., machines with serial numbers lower than E704…) to ROM 1 upon request.

ROM 1 requires System Software 2.0 or later, while ROM 3 requires System Software 5.0 or later. Although applications may work using older system software releases, they may not function properly due to the coordination of system software and ROM revisions.

## Changes from ROM 0 to ROM 1

### ADB

- Absolute ADB devices are now supported correctly.
- ADB fatal system error code is now $0911 instead of $0400.
- `ADBReset` routine now delays about 160 microseconds before reading the buttons.
- `ADBStatus TRUE` is now $FFFF instead of $0001.
- All ADB error codes now include the tool number.
- `SRQrmv` no longer crashes when you make the call with a command pending.

### AppleDisk 3.5

- AppleDisk 3.5 Macintosh block reads and writes now work as documented.
- Extended status call now returns bit 0 = 1 if AppleDisk 3.5 media has been switched since the last `READ`, `WRITE`, or `FORMAT`.
- New AppleDisk 3.5 status calls have been implemented to get internal variable and work buffer starting addresses.

### AppleTalk

- Link Access Protocol (LAP) inter-packet gap now handles added SCC delay.
- Name Binding Protocol (NBP) now considers uppercase and lowercase characters identical.
- A nonexistent protocol no longer hangs the dispatcher.

### Desk Manager

- `SaveScreen` and `RestoreScreen` now work.

### Event Manager

- Now auto-key events are not posted in the queue unless the queue is empty.
- `EMStartUp` and `EMShutDown` code has been optimized.
- Event Manager now returns an error instead of crashing when there is an attempt to post an invalid event.

### Integer Math

New Changes:
- Optimized the multiply routine.

RAM patches moved to ROM:
- Changes to `FixMul`, `FixRatio`, and `SDivide`.
- `SDivide` recovers from a divide by zero operation.
- New calls: `FracMul`, `FixDiv`, `FracDiv`, `FixRound`, `FracSqrt`, `FracCos`, `FracSin`, `FixATan2`, `HiWord`, `LoWord`, `Long2Fix`, `Fix2Long`, `Fix2Frac`, `Frac2Fix`, `Fix2X`, `Frac2X`, `X2Fix`, `X2Frac`.

### Memory Manager

- Optimized Purge and Compact for banks 0 and 1 and moved from RAM to ROM.
- RAM patches and enhancements moved to ROM.
- RAMdisk now returns bytes transferred count on DIB call.
- `SetHandleSize` makes a handle temporarily unpurgeable while changing handle size.

### Miscellaneous Tools

RAM patches and enhancements moved to ROM:
- `AbsClamp` fixes.
- Battery RAM routines work if data bank is set to a bank other than bank data is in.
- Firmware entry calls now return processor status in high byte instead of low byte.
- `GetAddr` with ref number $000E returns `SerFlag` address for SCC interrupts (useful if not using serial firmware).
- ID manager can reuse discarded IDs.

- Keyboard interrupts now enable VBL interrupts.
- `Munger` now works with 1-char strings and returns with A=0.
- New `SysBeep` call.
- `PackBytes` and `UnpackBytes` return with A=0.
- `ReadBParam` and `ReadBRAM` error codes corrected.
- `WriteBParam` and `WriteBRAM` do not return error codes (this is a documentation change).
- `WriteTimeHex` Bad Parameter error code is now $0301.

## Monitor

- 80-column screens maintained if break occurs and Pascal protocol in effect.
- AppleSoft tabbing in 80-column mode now works correctly.
- Control Panel's Maximum RAM Disk Size increased to 8128K instead of 4096K.
- Firmware version number returned is $1 instead of $0.
- Interrupts now disabled during paddle read routines.
- Interrupts re-enabled after fatal system error (for debug DAs).
- Mouse clamps with positive minimum and negative maximum works (e.g., $6000 min, $8000 max).
- New monitor command, pound sign (#), installs monitor entry and memory peeker classic desk accessories (unless already installed), accessible via the Control Panel. Reinstalled automatically on reset; disabled by power off only.
- New monitor command, Control-N, clears m, e, and x bits for native mode. (Control-R still switches to 8-bit, emulation mode.)
- `RESET` entry point at $00FA62 sets state register to $0C and shadow register to $08.
- Shadowing of the Super Hi-Res area in Bank 1 is no longer enabled automatically.
- `WAIT` routine now always exits with C=1.

## QuickDraw II

RAM patches and enhancements moved to ROM:

- 640-mode pen masks now work when `portRect` origin not a multiple of 8.
- Arcs, ovals, and round rects can be drawn across bank boundaries.
- Changes to round drawing routines: `PPToPort`, `GetFontLore`, `GetROMFont`, and `InflateTextBuffer`.
- Current bank bytes 100…106 no longer modified by scaling and mapping calls.
- `FontFlags` 1 and 2 added for pen width and color control.
- `FramePoly` returns with A=0.
- `GetPort` returns all four bytes of `GrafPort`.
- `HideCursor` and `ShowCursor` work correctly with obscured cursor.
- `MapRgn` now works on rectangular regions.
- Pixel painting routines support QuickDraw Auxiliary Tool Set stretching and shrinking.
- `PPToPort` now clips correctly to the current `portRect`.
- `QDStartUp` and `QDShutDown` save and restore the scan line interrupt vector.
- `RectInRgn` bug fixed.
- `ScrollRect` works when the `ClipRgn` and `VisRgn` are not rectangular.
- `SetSysFont` works.
- `StdPixels` now returns with A=0 if the pen is not visible.
- Text underline bug fixed.
- `TextBounds` works.

New QuickDraw changes:

- Busy flag now maintained correctly by `ClosePort`, `OffsetRgn`, `InsetRgn`, `KillPoly`, `FillRect`, `FrameOval`, `PaintOval`, `EraseOval`,

`InvertOval`, `FillOval`, `FrameArc`, `PaintArc`, `EraseArc`, `InvertArc`, `FillArc`, `FrameRRect`, `PaintRRect`, `EraseRRect`, `InvertRRect`, and `FillRRect`.

- Cursor appears in correct Super Hi-Res mode as determined by the low byte's bit 7 (320/640) of the `MasterSCB`.

**SANE**

- `Elems` now can be called from any part of memory.
- `HALT` exception jumping through the incorrect vector fixed.
- Integer overflow during conversion reported.
- `STATUS` call moved to ROM.

**Scheduler**

- Scheduler now accepts a flush function call.
- Task-handling RAM patch (on System Disk 1.0 and later) moved to ROM.

**Serial I/O**

- First character after an XON is no longer trashed when buffering is not enabled.
- If serial mode bit 17 = 1, parity and framing error suppression are defeated.
- Parity, baud, and data format commands work with buffering.
- `STATUS` call will not report that a character is ready if the character arrives with a parity or framing error.
- `STATUS` call works correctly with XON/XOFF protocol.

**SmartPort**

- `PR#5`, following a `PR#5` with I/O error (i.e., no disk in drive), now boots as expected.
- SmartPort manipulates only Slot 6 motor on detect so the IWM can run in fast mode.

**Sound**

- Fixed bug in `FFStopSound` call.
- Fixed low-level RAM read/write bug.
- Interrupts are disabled when the internal bell is active.
- Interrupts no longer need to be disabled when accessing sound RAM.
- New sound diagnostics with the following error codes: $0C001 = failed RAM data test, $0C002 = RAM address test, $0C003 = register data test, and $0C004 = control register test.
- Sound Manager RAM patches and enhancements moved to ROM.

**Text Tools**

RAM patches moved to ROM:

- RAM patches moved to ROM for `Writing` and `ErrorWriting` routines.
- `TextInit` Illegal device error now is in 16-bit mode instead of 8.

**Tool Locator**

- Optimized tool dispatcher.
- ROM tools present on a memory expansion card are installed.

## Changes from ROM 1 to ROM 3

ROM 3 is 256K (double the size of ROM 1) and contains several tools which do not exist in ROM 1. The patch file TS3 fixes known bugs in ROM 3 which were discovered after it was frozen. ROM 3 tools are basically System Software 5.0 tools, and the System Software 5.0 documentation covers these tools in detail. This Note only documents non-tool changes.

### AppleDisk 3.5 and SmartPort

- Use new routines for all block reads to fast RAM to eliminate double buffering.
- The extended DIB status call returns the device subtype byte $C1.
- Fixed anomalies described in SmartPort Technical Note #6, Apple IIGS SmartPort Errata.
- Fixed a ROM 1 bug that caused `Write Protected` to be returned with higher priority than `Device Offline` for the ProDOS `STATUS` call.

### AppleTalk

- AppleTalk moved to slots 1 and 2 from slot 7.

### Control Panel CDA

- The original Options menu is now the Keyboard menu and does not contain mouse parameters.
- A new Mouse menu is present. The new keyboard microcontroller allows finer control of mouse tracking, so a selection procedure better than yes or no is present. Parameters are also available to set the keyboard mouse feature, which allows the numeric keypad to emulate a mouse.
- Added an option to resize the RAM disk on the next reset in the RAM Disk menu. This option resets to No after one reboot and resizing so the RAM disk is not accidently reformatted on every boot thereafter.
- If slot 7 is set to AppleTalk, the Control Panel displays a warning if neither slot 1 nor slot 2 is similarly set.
- The Printer Port and Modem Port menus now display only those parameters that may be changed if AppleTalk is the selection for those ports.
- The RAM disk no longer has minimum and maximum settings, but rather one RAM disk size setting.

### Monitor

- Enhanced memory searching commands to automatically cross bank boundaries.
- Added Step and Trace debugging functions.
- Now provide vectors for the same functionality as the GS/OS System Service calls `MEMORY_MOVER`, `DYN_SLOT_ARBITER` and `SET_SYS_SPEED` in bank $E1.
- Now resize the RAM disk when the system is rebooted with the Control-Open Apple-Shift-Reset key combination.

- Handle text page 2 shadowing and power-up bits in the new CYA chip.
- Flash the border if the sound volume is set to zero and a beep is necessary.
- In ROM 1 and earlier, the Miscellaneous Tools mouse firmware called the 8-bit mouse routines in the $C400 space to do the work. In ROM 3, the 8-bit routines call the 16-bit routines to read the hardware. This change effectively means those programs which use 16-bit mouse calls (including desktop applications through the Event Manager) may use the mouse when slot 4 is set to Your Card.
- Slots 1 and 2 may now be set to Printer, Modem, AppleTalk, or Your Card. With System Software 5.0, slot 7 does not need to be set to AppleTalk to use an AppleTalk network, although one can do it for compatibility. There is no transparent printing firmware in slot 7.
- The Alternate Display Mode CDA no longer sets the system to fast speed when normal speed is selected in the Control Panel.
- Added a new command, `{val}=V`, to set the video screen display I/O switches when resuming a program.
- Control-T command now works as a toggle—executing it once changes to text mode, but now executing it again switches back to the previous video mode You may change this saved video mode with the `=V` command.
- Battery RAM value $59 now controls the presence of the Visit Monitor and Memory Peeker CDAs. If this byte has the high bit set at boot time, the CDAs are automatically installed.
- The Monitor and Memory Peeker both allow the use of Control-X to terminate a long display (i.e., a handle list or memory dump).

**Serial I/O**

- XON and XOFF are no longer sent with the high bit set when buffering is enabled.
- Terminal mode cursor is more consistent with the rest of the system.
- Extended Interface calls now return errors in the carry and the accumulator.

**Toolbox**

The following tools are now in ROM:

- Window Manager
- Menu Manager
- Control Manager
- Line Edit
- Dialog Manager
- Scrap Manager
- Font Manager
- List Manager

**Further Reference**

- *Apple IIGS Firmware Reference*

- *Apple IIGS Toolbox Reference*
- Apple IIGS Technical Note #52, Loading and Special Memory
- SmartPort Technical Note #6, Apple IIGS SmartPort Errata



# Apple IIGS
# #27: Graphics Image File Formats

Revised by: Matt Deatherage November 1988
Written by: Steve Glass, Eagle Berns, Art Cabral,
Pete McDonald & Rilla Reynolds October 1987

This Technical Note formerly described the file formats for Apple IIGS graphics image files. File formats are now documented in Apple II File Type Notes under corresponding file types and auxiliary types:

---

**File Type $C0**

Auxiliary Type $0000 "PaintWorks" Packed Format
Auxiliary Type $0001 PackBytes Packed Format
Auxiliary Type $0002 "Apple Preferred" Packed Format

**File Type $C1**

Auxiliary Type $0000 32K unpacked picture image
Auxiliary Type $0001 Unpacked QuickDraw II picture

**Further Reference**

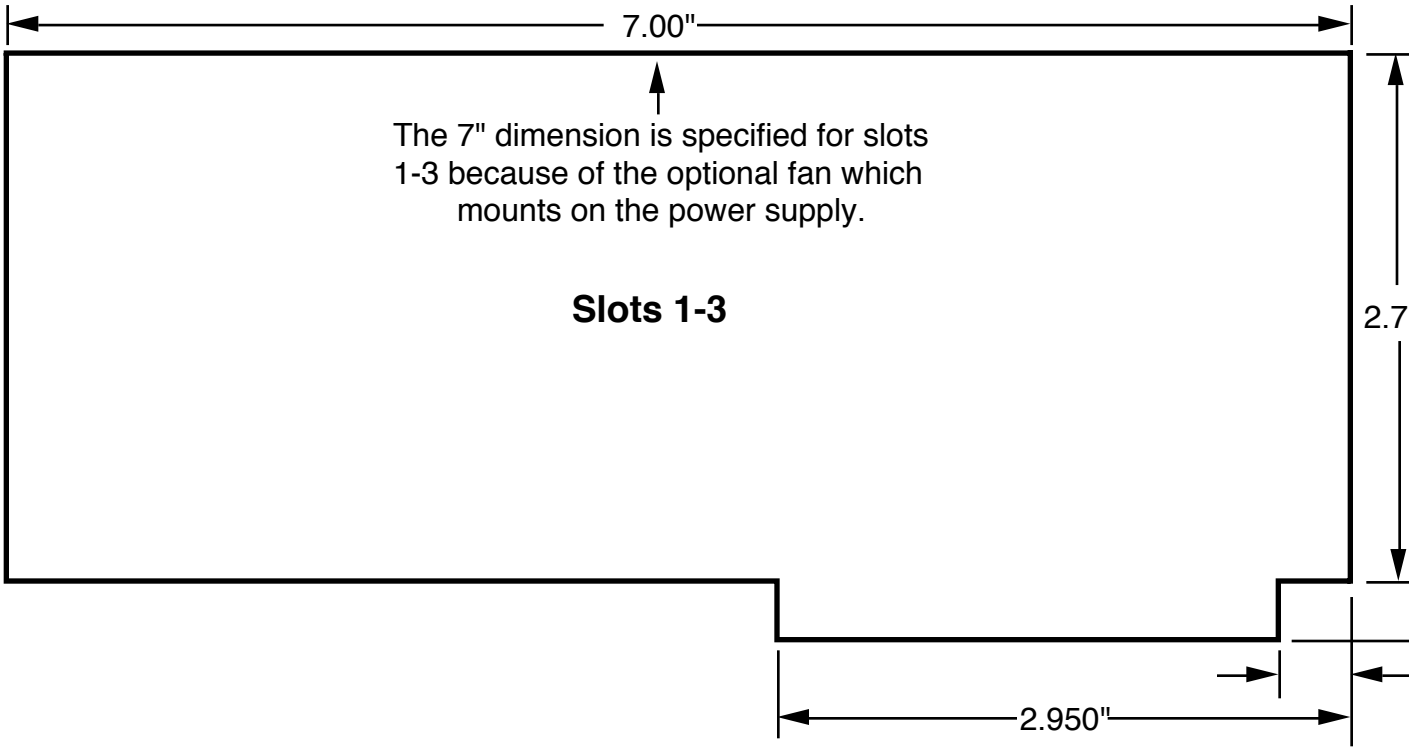
- Apple II File Type Notes

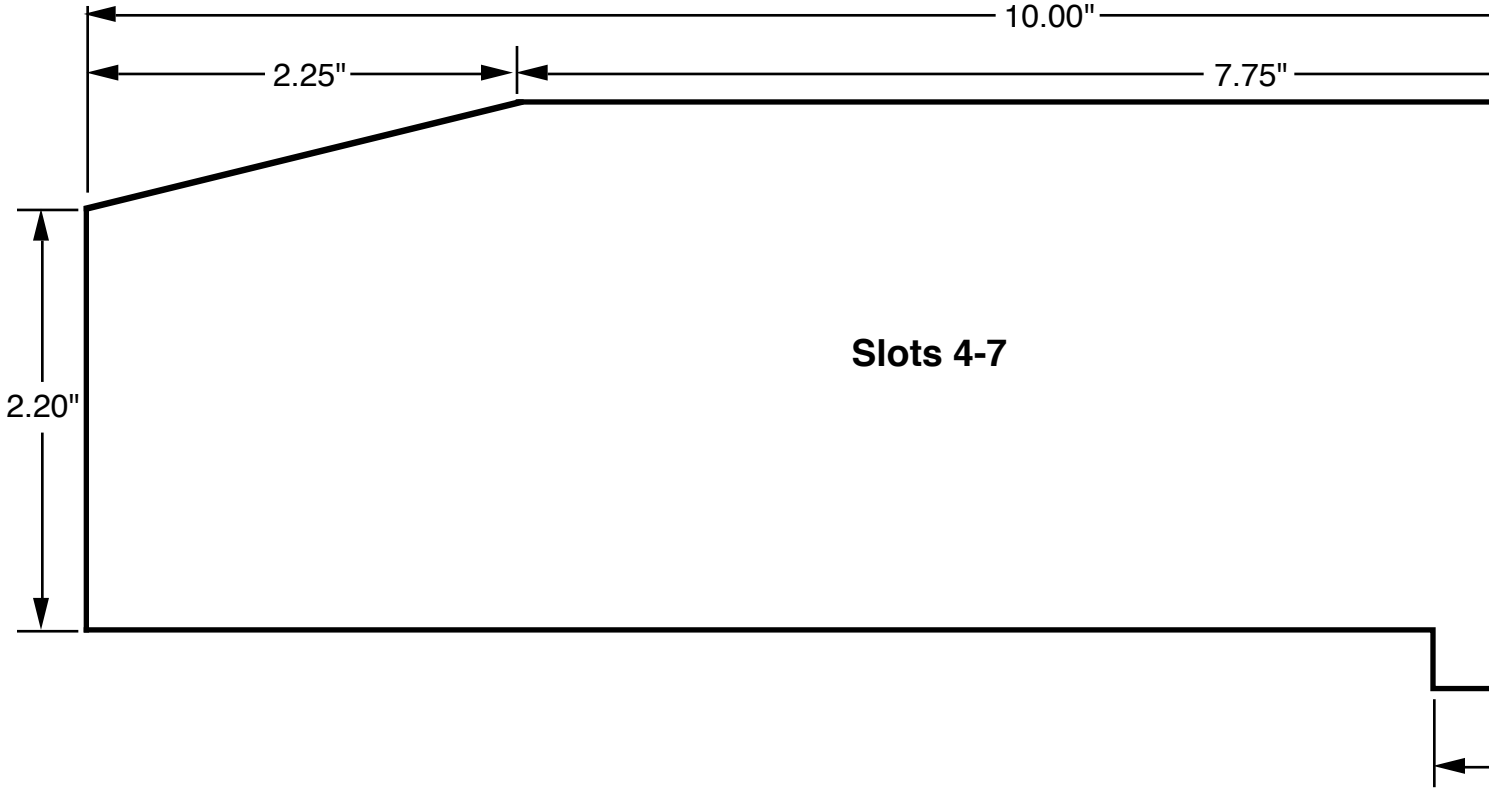


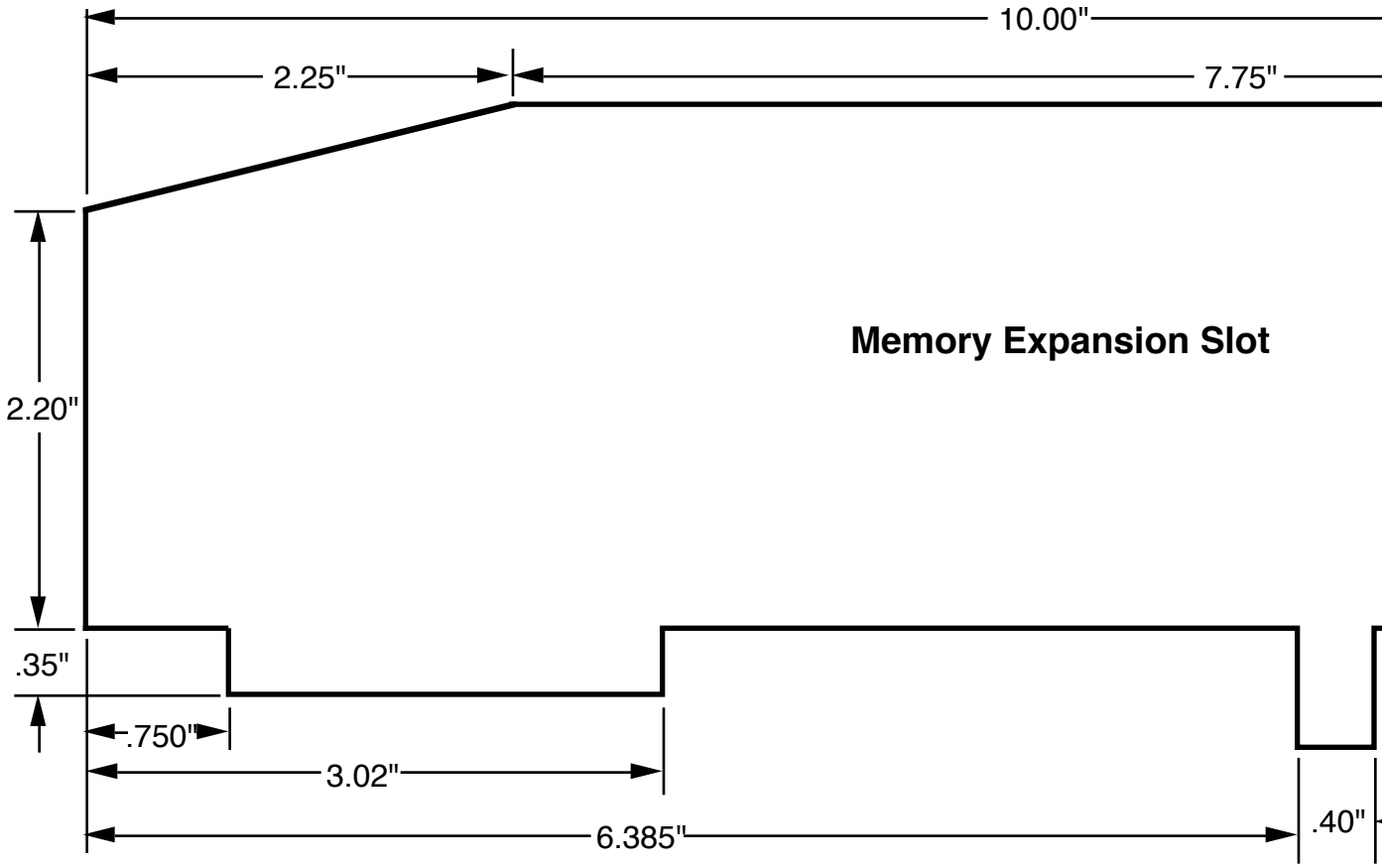
# Apple IIGS
# #28: Interface Card Design Guidelines

Revised by: Matt Deatherage November 1988
Written by: Cameron Birse October 1987

This Technical Note describes suggested dimensions for interface cards for the Apple IIGS and Apple IIe upgraded sytems.


7.00"
The 7" dimension is specified for slots
1-3 because of the optional fan which
mounts on the power supply.
Slots 1-3
2.7
2.950"


10.00"
2.25"
7.75"
2.20"
Slots 4-7


10.00"
2.25"
7.75"
Memory Expansion Slot
2.20"
.35"
.750"
3.02"
6.385"
.40"



# Apple IIGS
# #30: *Apple IIGS Hardware Reference* Updates

Revised by: Jim Luther September 1990
Written by: Rilla Reynolds & Jim Luther October 1987

This Technical Note includes updates to the *Apple IIGS Hardware Reference*, published by Addison-Wesley. Please contact Apple II Developer Technical Support at the address listed in Apple II Technical Note #0 if you have additional corrections or suggestions for these manuals.

**Changes since July 1990:** Changed the description in "Signals at the Serial Ports and the Serial Communications Controller" to correctly note that the SCC can support a maximum asynchronous transmission rate of 57,600 bits per second (bps) in X16 clock mode.

---

There are two editions of the *Apple IIGS Hardware Reference*, the first edition (July 1987) which covers the original Apple IIGS only, and the second edition (1989) which covers both original Apple IIGS and the 1 MB Apple IIGS. Because page numbers have changed between the two editions and because an update to one edition may not be needed in both editions, this Note organizes corrections by chapter, always noting corrections to the Second Edition followed by corrections to the First Edition.

## Chapter 3: Memory

**Second Edition—**Page 40, Table 3-2, Bits in the State register
**First Edition—**Page 36, Table 3-2, Bits in the State register

Switch the given values and descriptions for bits 7 and 2 as follows:

| Bit | Value | Description |
|---|---|---|
| 7 | 1 | `ALTZP`: If this bit is 1, then bank-switched memory, stack, and direct page are in *auxiliary* memory. |
| | 0 | If this bit is 0, then bank-switched memory, stack, and direct page are in *main* memory. |
| 2 | 1 | `LCBNK2`: If this bit is 1, language-card RAM *bank 2* is selected. |
| | 0 | If this bit is 0, language-card RAM *bank 1* is selected. |

## Chapter 6: The Apple Desktop Bus

**Second Edition—**Page 148, after final paragraph

Add a new heading and description:

### Control Panel Control Jumper

The ADB microcontroller provided with the 1 MB Apple IIGS includes an input that disables the text Control Panel (normally available via the Classic Desk Accessory menu). This feature allows the system parameters to be set and then protected from changes made via the text Control Panel. A jumper across the pins of connector `S1` removes the text Control Panel from the Classic Desk Accessory menu. All other installed classic desk accessories are still available in the Classic Desk Accessory menu when the `S1` jumper is installed. The `S1` connector is located near the ADB microcontroller at motherboard location `F12`.

**Note:** The `S1` jumper does not prevent the system parameters from being changed with the graphic Control Panel (a new desk accessory normally available from the Apple menu of the Finder or of any other application that includes the Apple menu).

**First Edition—**Page 130, Table 6-9, Command byte syntax

The first row in the table should read:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| x | x | x | x | 0 | 0 | 0 | 0 | Send Reset |

and not

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $A_3$ | $A_2$ | $A_1$ | $A_0$ | 0 | 0 | 0 | 0 | Device Reset |

**First Edition—**Page 131, Device Reset

Replace “Device Reset” with “Send Reset.” The paragraph should be: “When a device receives a *Send Reset* command, it will clear all pending operations and data, and will initialize to the power-on state. *The Send Reset command is not device-specific; it is sent to all devices simultaneously*.”

**First Edition—**Pages 138-139, Collision detection

The fourth sentence in the last paragraph should be: “By using the Listen register 3 command, the host can move *the device with the activator pressed*.”

## Chapter 7: Built-In I/O Ports and Clock

**Second Edition—**Page 154, Table 7-3, Disk-port soft switches
**First Edition—**Page 146, Table 7-3, Disk-port soft switches

$C0E8 *Drive disabled*
$C0E9 *Drive enabled*
$C0EA *Drive 1* select
$C0EB *Drive 2* select

In addition to the corrections listed for Table 7-3, the reference to "spindle motor switches" in the paragraph following the table should be replaced with "drive enable switches."

**Second Edition—**Page 155, Table 7-4, IWM states
**First Edition—**Page 146, Table 7-4, IWM states

Change the table to the following:

| Q7 | Q6 | Drive | Operation |
|---|---|---|---|
| 0 | 0 | enabled | Read Data register |
| 0 | 1 | - | Read Status register |
| 1 | 0 | - | Read Handshake register |
| 1 | 1 | disabled | Write Mode register |
| 1 | 1 | enabled | Write Data register |

1 = asserted state 0 = negated state - = do not care

**First Edition—**Page 146, after Table 7-4, IWM states

The following text and table should also be added:

"The drive enable switches and the drive select switches control the state of the disk port signals DR1 and DR2. The following table shows the relationship between these."

| Soft Switches | | | | Disk Port Signals | |
|---|---|---|---|---|---|
| $C0E8 | $C0E9 | $C0EA | $C0EB | DR1 | DR2 |
| 1 | - | - | - | 0 | 0 |
| - | 1 | 1 | - | 1 | 0 |
| - | 1 | - | 1 | 0 | 1 |

1 = asserted state 0 = negated state - = do not care

**First Edition—**Page 147, The Mode register

The IWM Mode register is a write-only register, so disregard the advice to use only a read-modify-write instruction sequence when manipulating bits.

**Second Edition—**Pages 156-7, Table 7-5, Bits in the Mode register
**First Edition—**Pages 147-8, Table 7-5, Bits in the Mode register

For **Second Edition**, change the description for bit 2, value 0 as shown. For **First Edition**, switch the given values and descriptions for bits 1, 2, and 4 as shown.

| Bit | Value | Description |
|---|---|---|
| 4 | 1 | 8-MHz read-clock speed selected. |
| | 0 | 7-MHz read-clock speed selected. Set to 0 for all Apple IIGS disk accesses. |
| 2 | 1 | 1-second timer is not selected. |
| | 0 | 1-second timer selected. When the current disk drive is deselected, the drive will remain enabled for 1 second if this bit is *clear*. |
| 1 | 1 | Asynchronous handshake protocol selected; for all except 5.25-inch Apple disk drives. |
| | 0 | Synchronous handshake protocol selected; for 5.25-inch Apple disk drives. |

**Second Edition—**Page 159, The serial ports
**First Edition—**Page 150, The serial ports

The first sentence should read: "The Apple IIGS has two serial ports located at the back of the computer, which may provide synchronous and asynchronous serial communications."

**Second Edition—**Page 160, Table 7-9, Pins on a serial-port connector
**First Edition—**Page 151, Table 7-8, Pins on a serial-port connector

Replace the table title and table with this table title, table and note:

**Table 7-x** Signal assignments for the mini 8-pin serial port connectors

| Pin Number | Signal name | Signal Description |
|---|---|---|
| 1 | HSKo | Handshake output. Driven uninverted from the SCC's /DTR output.<br>Voh = 3.6V; Vol = -3.6V; Rl - 450Ω |
| 2 | HSKi | Handshake input or external clock. Received inverted at SCC's /CTS and /TRxC inputs.<br>Vih = 0.2V; Vil = -0.2V; Ri = 12KΩ |
| 3 | TxD- | Transmit data (inverted). Driven inverted from SCC's TxD output; tri-stated when SCC's /RTS is not asserted.<br>Voh = 3.6V; Vol = -3.6V; Rl = 450Ω |
| 4 | GND | Signal ground. Connected to logic and chassis ground. |
| 5 | RxD- | Receive data (inverted). Received inverted at SCC's RxD input.<br>Vih = 0.2V; Vil = -0.2V; Ri = 12KΩ |
| 6 | TxD+ | Transmit data. Driven uninverted from SCC's TxD output; tri-stated when SCC's /RTS is not asserted.<br>Voh = 3.6V; Vol = -3.6V; Rl = 450Ω |
| 7 | GPi | General-purpose input. Received inverted at SCC's /DCD inputs.<br>Vih = 0.2V; Vil = -0.2V; Ri = 12KΩ |
| 8 | RxD+ | Receive data. Received uninverted at SCC's RxD input.<br>Vih = 0.2V; Vil = -0.2V; Ri = 12KΩ |

**Note:** Absolute values of specified voltages are minimums; Ri is a minimum, Rl is a maximum.

**Second Edition—**Page 164, after Figure 7-9
**First Edition—**Page 155, after Figure 7-9

Add a new heading and description:

**Signals at the Serial Ports and the Serial Communications Controller**

The Apple IIGS has two serial ports which are compatible with most RS-232-C devices. This section describes the input and output signals provided at the serial ports. This section also discusses some input signals to the 8530 Serial Communications Controller (SCC) chip that are not described in the *Apple IIGS Hardware Reference*.

The transmit-data and receive-data lines of the Apple IIGS serial interface conform to the EIA standard RS-422, which differs from the more commonly used RS-232-C standard in that, whereas an RS-232-C transmitter modulates a signal with respect to a common ground, an RS-422 transmitter modulates the signal against an inverted copy of the same signal (to generate a differential signal). The RS-232-C receiver senses whether the received signal is sufficiently negative with respect to ground to be logical 1, whereas the RS-422 receiver simply senses which line is more negative than the other. An RS-422 signal is therefore more immune to noise and interference, and degrades less over distance, than an RS-232-C signal. If you ground the positive side of each RS-422 receiver and leave unconnected the positive side of each transmitter, you have essentially converted to EIA standard RS-423, which can be used to communicate with most RS-232-C devices over distances up to fifty feet, as illustrated in Figures 7-x1 and 7-x2.

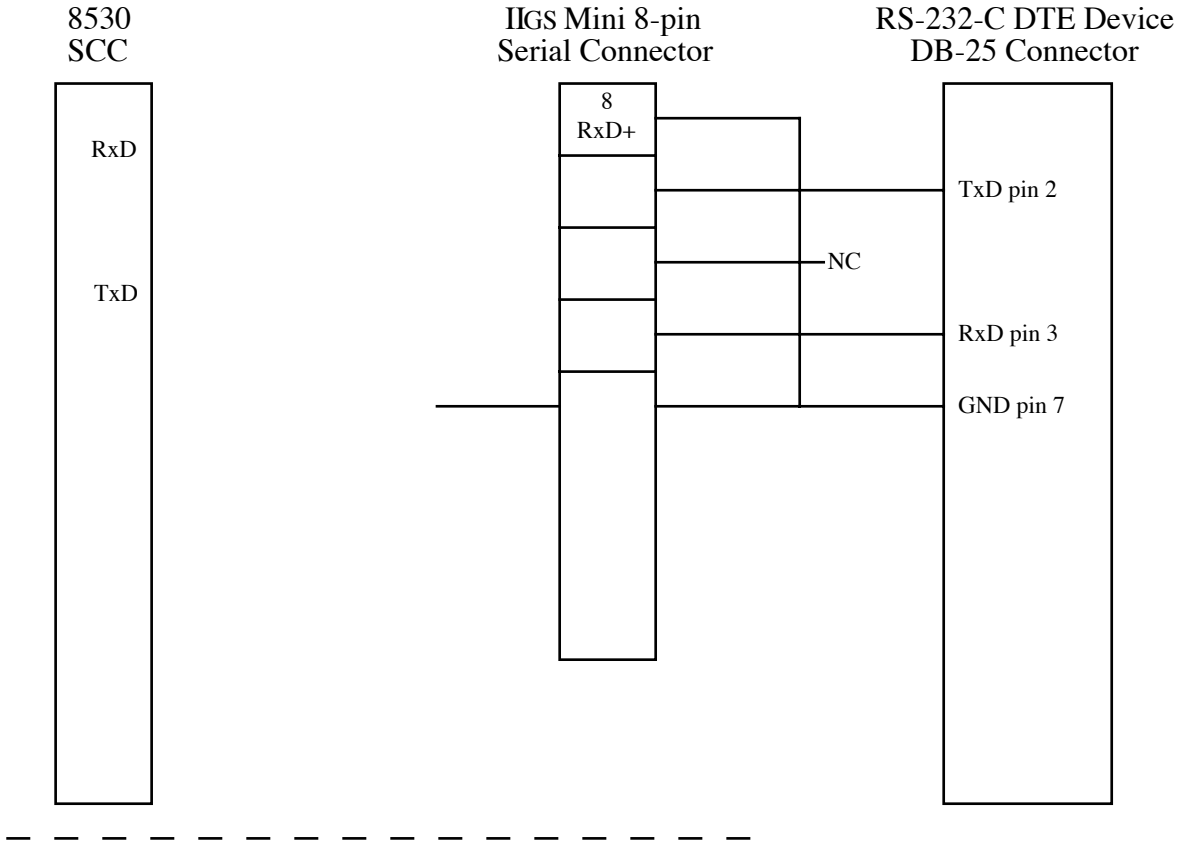



**Figure 7-x1–Apple IIGS Connection to an RS-232-C DTE Device**

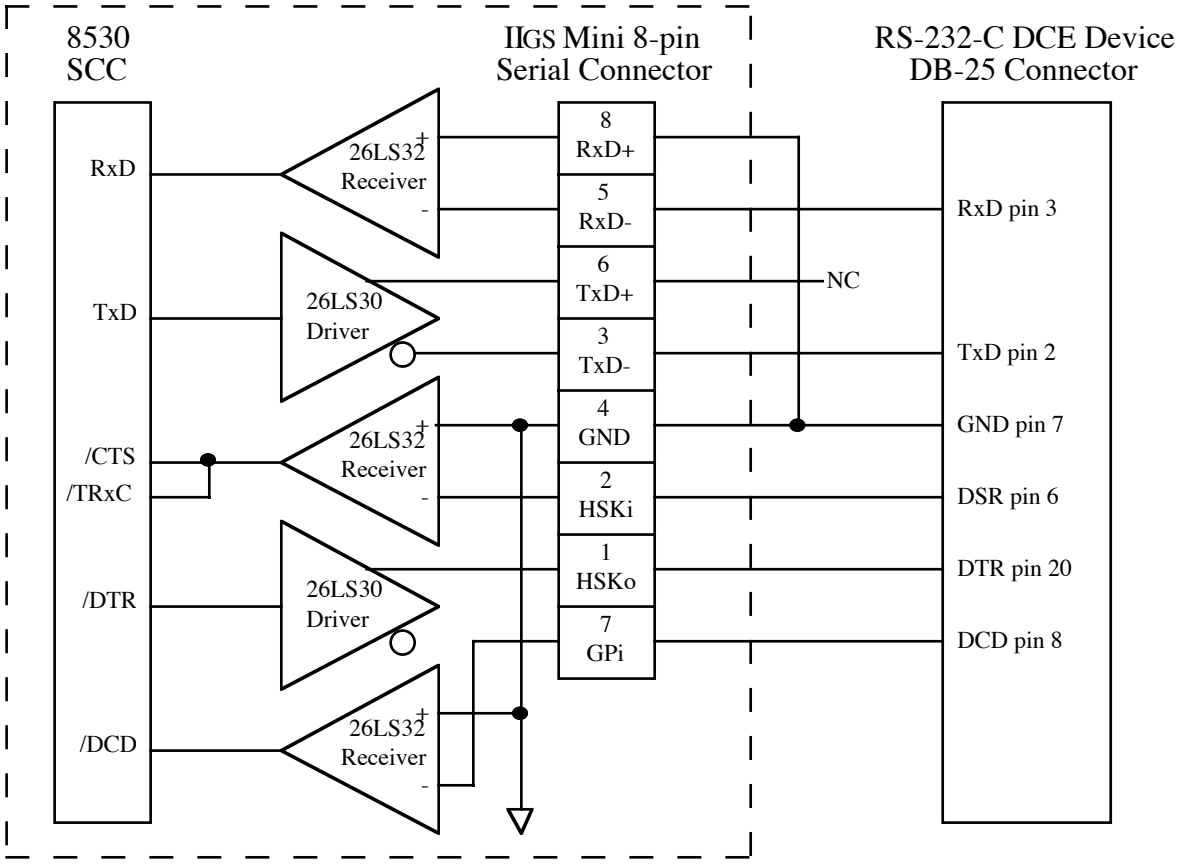



**Figure 7-x2–Apple IIGS Connection to an RS-232-C DCE Device**

The serial inputs and outputs of the SCC are connected to the external connectors through differential line drivers (26LS30) and receivers (26LS32). The output line drivers are tri-state devices and can be put in the high-impedance mode between transmissions to allow other devices (i.e., AppleTalk devices) to transmit over those lines. A line driver is activated by lowering the SCC's Request To Send (/RTS) output for that port.

The Handshake Output signal (HSKo, pin 1) for each Apple IIGS serial port originates at the SCC's /DTR output for that port and is driven uninverted by an RS-422 line driver (26LS30). Each port's Handshake Input signal (HSKi, pin 2) is received and inverted through a differential receiver (26LS32). The output of the differential receiver is connected to the SCC's Clear To Send (/CTS) and Transmit/Receive Clock (/TRxC) inputs for that port. HSKi is designed to accept an external device's Data Terminal Ready (DTR) handshake signal through the /CTS input. The /CTS input to the SCC can be polled by software or can be used to generate an interrupt. The HSKi line is connected to the SCC's Transmit/Receive Clock (/TRxC) input for that port, so that an external device can perform high-speed synchronous data exchange. Note that you can't use the HSKi line for receiving DTR if you're using it to receive a high-speed data clock.

Each port's General-Purpose input (GPi, pin 7) is received and inverted through a differential receiver (26LS32). The output of the differential receiver is connected to the SCC's Data Carrier Detect (/DCD) input for that port. This input can be used to provide a handshake

signal from an external device to the computer. The /DCD input to the SCC can be polled by software or can be used to generate an interrupt.

**Note:** Because a 26LS32 differential receiver is used for the external handshake or clock signals to the SCC, the signals must be bipolar, alternating between a positive voltage and a negative voltage with respect to the internally grounded input. If a device uses ground (0 volts) as one of its handshake logic levels, the receiver interprets that level as an indeterminate state, with unpredictable results.

The SCC's Receive/Transmit Clock (/RTxC) inputs for both ports are driven by a single crystal oscillator circuit. This is accomplished by connecting a 3.6864 MHz crystal between the /RTxC and Synchronization (/SYNC) input of port A. Port B's /RTxC pin is connected to port A's /SYNC pin to drive port B's clocks from port A's oscillator circuit. Because of this single circuit, Write Register 11 (WR11) bit 7 must be set to 1 for SCC port A and must be set to 0 for SCC port B. The SCC itself is clocked at 3.58 MHz by the Apple IIGS' Color-Reference clock (CREF) at the SCC's PCLK clock input. The maximum asynchronous transmission rate supported by the SCC is 57,600 bits per second (bps) in X16 clock mode (WR4=01xxxxxx).

The SCC's Interrupt Enable In (IEI) and Interrupt Acknowledge (/INTACK) inputs are both tied to logical high in the Apple IIGS. Keeping the SCC's IEI input high enables the SCC to always generate interrupts if interrupt modes are enabled through software. Keeping the SCC's /INTACK input high leaves the SCC in Interrupt Without Acknowledge interrupt mode.

## Chapter 8: I/O Expansion Slots

**First Edition**—Page 167, Direct memory access

DMA bank register location is $C037.

**Further Reference:**

- *Apple IIGS Hardware Reference,* both editions



# Apple IIGS
# #31: Redirecting Output in APW C

Revised by: Guillermo Ortiz November 1988
Written by: Guillermo Ortiz November 1987

This Technical Note presents a sample program which shows how to send output to different devices under the Apple Programmer's Workshop (APW) shell.

---

Many programmers find the ability to redirect output an expecially useful feature. The following is a sample C program which allows this redirection through an APW shell command. Note that this is not applicable to MPW IIGS C since it is not part of the APW environment.

```
/*
redirect.c
Testing the shell REDIRECT command within APW C
Demonstrates how to send the output to different devices,
a disk file, the printer, and then back to the screen
last modified by Guillermo Ortiz 09/21/87

NOTE: This program checks no errors whatsoever. It expects to
be able to open the file with no problems and expects the
printer to be readily available.

Also remember that for this test to work the file has to be of
the type 'EXE' (executable from the shell only.)
*/

#include <types.h>
#include <misctool.h>
#include <stdio.h>
#include <shell.h>
#include <string.h>

char timestrg[20];          /* string to store the ascii time */
char myfile[80];            /* string to store the filename */
char str[80];               /* dummy string */
int dev=0x0001;             /* standard output */
int app=0x0000;             /* app=0 file is deleted, other will append */

PrintToFile()
{
 printf("Please enter the output filename: \n");
 gets(myfile);
 if (strlen(myfile)==0)
   {
  printf("Error in entering the filename, quit.\n");
  exit(0);
}
```

```
        /* REDIRECT call requires pascal string */
        c2pstr(myfile);

       /*  use  the  REDIRECT  shell  command  to  redirect  the  file output  to  the
*/
        REDIRECT(dev, app, myfile);

        /* now print a few lines of text */
        printf("This is my first line of text.\n");
        printf("And this is the second line.\n");
        printf("Finally the third and last line of text.\n");

       }

       PrintToPrinter()
       {
        /* now redirect to output to the .PRINTER. */
        REDIRECT(dev, app, "\010.PRINTER.");

        printf("We should now be going to the printer.\n");
        ReadAsciiTime(timestrg);
        printf ("The time now is %s\n",timestrg);
       }

       BackToScreen()
       {
        /* Last REDIRECT the output back to the screen. */
        REDIRECT(dev, app, "\010.CONSOLE.");

        printf("The testing of REDIRECTing the output is done.\n");
        ReadAsciiTime(timestrg);
        printf ("The time now is %s\n",timestrg);
       }

       main()
       {
        ReadAsciiTime(timestrg);
        printf ("The starting time is %s\n",timestrg);

        PrintToFile();
        PrintToPrinter();
        BackToScreen();
       }
```

## Further Reference

- *Apple IIGS Programmer's Workshop Reference*
- *Apple IIGS Programmer's Workshop C Reference*



# Apple IIGS
# #32: /INH Line Anomaly

Revised by: Glenn A. Baxter & Rob Moore November 1988
Written by: Glenn A. Baxter December 1986

This Technical Note describes a hardware anomaly which affects the use of the /INH line on the Apple IIGS.

---

The Apple IIGS maps logical addresses in main and auxiliary RAM spaces to physical RAM devices in such a way that using the /INH line can cause bus contention under certain conditions. This Note describes the problem and suggests a solution strategy.

In the Apple IIGS, main memory resides within four physical 64 x 4 DRAMs. Memory is logically mapped into two separate banks of 64K x 8. The logical map of main memory is slightly different than what one might expect. Owing to the demands of new video modes on the IIGS, the DRAMs need a greater amount of time to perform their function. The easiest way to allocate time in a fixed, time-based system is to use a memory interleaving mechanism, and the IIGS implements its video in this fashion.

As a result of this interleaving scheme, the logical map of main and auxiliary memory does not correspond directly to physical DRAMs, but are split in three places. The split looks like the following:

| First Physical 64K | Second Physical 64K | | |
|---|---|---|---|
| Main Memory | $0000 – $5FFF | Auxiliary Memory | $0000 – $5FFF |
| Auxiliary Memory | $6000 – $9FFF | Main Memory | $6000 – $9FFF |
| Main Memory | $A000 – $FFFF | Auxiliary Memory | $A000 – $FFFF |

Only part of the first physical bank of RAM is inhibited when /INH is brought low; therefore, the /INH function only works between $0000 – $5FFF and $A000 – $FFFF in main memory and $6000 – $6FFF in auxiliary memory. If a card attempts to inhibit main memory in the range of $6000 – $9FFF or auxiliary memory in the ranges $0000 – $5FFF or $A000 – $FFFF, bus contention results as both the Mega II and the 74HCT245 buffer device attempt to drive the bus simultaneously (which can damage the Mega II).

Because earlier Apple II systems do not arrange their physical memory as described above, cards which use the /INH line may be compatible with the Apple ][+ and IIe, but not with the IIGS. To be compatible with all Apple II systems, a card should include an address mask that will prevent /INH from going low when the address in in the sensitive ranges of main or auxiliary memory.

The following logic equation represents an appropriate blocking signal for main memory inhibition:

```
BLOCK  =      /A15   *      A14    *      A13    ;BLOCK $6000-$7FFF
       +      A15    *      /A14   *      /A13   ;BLOCK $8000-$9FFF
```



# Apple IIGS
# #33: ERRORDEATH Macro

Revised by: Jim Mensch & Matt Deatherage November 1988
Written by: Allan Bell, Apple Australia & Jim Merritt December 1987

This Technical Note presents a short macro which an assembly language program can invoke to handle fatal error conditions.

---

Early versions of Apple-approved sample assembly language code for the Apple IIGS often invoked an APW macro named `ERRORDEATH`. This macro generated code that was appropriate for handling situations where program execution simply could not proceed due to "fatal" errors, such as a failure to load one or more tools that are required to display more sophisticated error dialogs or the inability to allocate sufficient direct page space for essential tool sets. The macro libraries of prototype APW systems included `ERRORDEATH`, but the release version does not to promote the use of more sophisticated error handling techniques in commercial software packages. The MPW IIGS release never included `ERRORDEATH`.

Below are two versions of `ERRORDEATH`; one is compatible with official standard releases of APW and the other with MPW IIGS. While Apple recommends avoiding the use of `ERRORDEATH` in software intended for commercial release, we feel the code is still useful for providing minimal error handling capability in prototype code and a brief, yet sophisticated, example of macro construction.

**APW Assembler version:**

```
          MACRO
&lab          ERRORDEATH &text
&lab          bcc end&syscnt
              pha
              pea x&syscnt|-16
              pea x&syscnt
              ldx #$1503
              jsl $E10000
x&syscnt      dc i1'end&syscnt-x&syscnt-1'
              dc c"&text"
              dc i1'13',i1'13'
              dc c'Error was $'
end&syscnt    anop
          MEND
```

**MPW IIGS Assembler version:**

```
          MACRO
          ErrorDeath &text
          bcc @EDeathEnd
             pha
          pea @Message>>16
          pea @Message
          ldx #$1503
          jsl $E10000
@Message  dc.B      @EDeathEnd-@Message-1
          dc.B      &text
          dc.B      13
          dc.B      'Error Was $'
@EDeathEnd
          MEnd
```

The "active ingredient" in the `ERRORDEATH` macro is the call to `SysFailMgr` ($1503), which is made if carry is set at the time control passes to the beginning of the expanded macro code sequence. The APW and MPW IIGS assembler macro expansion mechanisms insert the value represented by the character string argument marker, `&text`, into the generated code stream and

provide SysFailMgr with a pointer to that string. The pseudo-argument, &syscnt, generates unique labels in the positions occupied by the expressions x&syscnt and end&syscnt, which makes it possible to invoke ERRORDEATH more than once during any particular source assembly. In the MPW IIGS version of the macro, the MPW IIGS assembler creates a unique label for any label beginning with the at sign (@), effectively doing the equivalent of the &syscnt in the APW version.

To use ERRORDEATH, simply invoke it after any code sequence or subroutine call that sets the carry when it encounters an error (clears it, otherwise) and leaves an appropriate error code in the accumulator. Note that all ProDOS and Toolbox calls observe this convention. When control passes to the beginning of the ERRORDEATH code sequence, the CPU should be in full-native mode, which means the emulation bit should be clear and the accumulator and index registers should be 16-bits wide). Here is a small code segment which demonstrates invoking the macro:

```
        pushword #21                ; Dialog Manager
        pushword #0                 ; Use any version
        _LoadOneTool

; If carry is now SET, following macro terminates program execution
; with the "sliding Apple" error screen.

IfWeGoofed    ERRORDEATH 'Cannot load Dialog Manager!'

; *** If no error, normal execution continues here ***
```



# Apple IIGS
# #34: Low-Level QuickDraw II Routines

Revised by: Dave "Evad Snoyl" Lyons, C.K. Haun, Keith Rollin,
Steven Glass, Matt Deatherage & Eric Soldan January 1991
Written by: Steven Glass May 1988

This Technical Note describes the low-level routines which QuickDraw II uses to do much of the work in standard calls and mechanisms for calling these routines and accessing their data.

**Changed since November 1990:** Added a Note on custom bottleneck procedures and updated information on `ShieldCursor` and `UnShieldCursor`.

---

QuickDraw II lets you customize low-level drawing operations by intercepting the "bottleneck procedures." QuickDraw II calls an appropriate "bottleneck proc" every time it receives a call to draw an object, measure text, or deal with pictures. For example, if an application calls `PaintOval`, QuickDraw II calls `StdOval` to do the real work, and if an application calls `InvertRgn`, QuickDraw II calls `StdRgn` to do the work.

Installing your own bottleneck procedures is a little bit tricky. The QuickDraw II `SetStdProcs` call accepts a pointer to a 56-byte ($38 hex) record and fills that record with the addresses of the standard bottleneck procedures of QuickDraw II. You may modify this record by replacing those addresses with the addresses of your own custom bottleneck procedures minus one. (QuickDraw II pushes the address on the stack and executes an `RTL` to it, so the address in the record must point to the byte before the routine.)

**Note**: A custom bottleneck procedure must not begin at the first byte of a segment. If it does, then the segment could load at the beginning of a bank, and the address minus one would be in the wrong bank and `RTL` would transfer control to the wrong location. (See Apple IIGS Technical Note #90, 65816 Tips and Pitfalls.)

After installing your own procedures, you use `SetGrafProcs` to tell QuickDraw II about them. The format of this call is as follows (taken from the E16.QUICKDRAW file in APW):

```
ostdText      GEQU   $00 ; Pointer - QDProcs -
ostdLine      GEQU   $04 ; Pointer - QDProcs -
ostdRect      GEQU   $08 ; Pointer - QDProcs -
ostdRRect     GEQU   $0C ; Pointer - QDProcs -
ostdOval      GEQU   $10 ; Pointer - QDProcs -
ostdArc       GEQU   $14 ; Pointer - QDProcs -
ostdPoly      GEQU   $18 ; Pointer - QDProcs -
ostdRgn       GEQU   $1C ; Pointer - QDProcs -
ostdPixels    GEQU   $20 ; Pointer - QDProcs -
ostdComment   GEQU   $24 ; Pointer - QDProcs -
```

```
ostdTxMeas    GEQU   $28 ; Pointer - QDProcs -
ostdTxBnds    GEQU   $2C ; Pointer - QDProcs -
ostdGetPic    GEQU   $30 ; Pointer - QDProcs -
ostdPutPic    GEQU   $34 ; Pointer - QDProcs -
```

The following code fragment shows how you might replace the `StdRect` procedure with your own for a given window:

```
pha                              ; open a test window
pha
PushLong #MWindData              ; standard setup for NewWindow
_NewWindow
_SetPort

PushLong #MyProcs                ; get a record to modify
_SetStdProcs

ldy #ostdRect                    ; get the low word of my rectangle routine
lda #myRect-1                    ; (minus one) and patch it in to the record
sta myProcs,y
lda #^myRect                     ; do the same for the high word
sta myProcs+2,y

PushLong #MyProcs                ; install the procs
_SetGrafProcs
```

The interface to bottleneck procedures is different from the interface to other QuickDraw II routines; you do not make calls via the tool dispatcher and you pass most parameters on the direct page and in registers (rather than on the stack). To write your own bottleneck procedures, you have to know where the inputs to each call are kept and how to call the standard procedures from inside your own procedures.

The standard bottleneck procedures are accessed through vectors in bank $E0.

```
StdText     $E01E04
StdLine     $E01E08
StdRect     $E01E0C
StdRRect    $E01E10
StdOval     $E01E14
StdArc      $E01E18
StdPoly     $E01E1C
StdRgn      $E01E20
StdPixels   $E01E24
StdComment  $E01E28
StdTxMeas   $E01E2C
StdTxBnds   $E01E30
StdGetPic   $E01E34
StdPutPic   $E01E38
```

When you call any of the standard procedures, the first direct page of QuickDraw II is active. If you pass variables on any direct page other than the first (direct page locations greater than $FF), you can use a simple trick to access them. For example, to access `TheFillPat` ($10E) without changing the direct page register:

```
ldx    #$100                     ;offset to second DP
lda    >$0E,X                    ;gets "DP" location $10E
```

Certain locations on the direct page are always valid:

| | |
|---|---|
| `PortRef` | $24 |
| `MaxWidth` | $28 |
| `MasterSCB` | $08 |
| `UserID` | $0A |

DrawVerb is usually valid, but not always:

```
DrawVerb   $38
```

Each of the bottleneck procedures uses the direct page differently.

QuickDraw II has an interesting bug relating to the standard conic bottleneck procedures. If you replace any of the standard procedures with your own, QuickDraw II does not perform some of the setups it normally would before calling the standard conic procedures (`stdRRect`, `stdOval`, `stdArc`). For example, if you replace StdRect with a custom rectangle routine, but leave the other conic pointers alone (as shown in the code fragment above), QuickDraw II will not do all of the normal setups when calling the standard conic routines. To deal with this bug of QuickDraw II, you must patch out the additional bottleneck procedures and set up those direct pages locations yourself, or the results will not be what you expect. The QuickDraw II direct-page variables you must initialize yourself in this instance are bulleted (•) below.

**StdText**

| | | |
|---|---|---|
| `DrawVerb` | $38 | Describes the kind of text to draw. There are three possible values:<br>DrawCharVerb 0<br>DrawTextVerb 1<br>DrawCStrVerb 2 |
| `TextPtr` | $DA | If the draw verb is DrawTextVerb or DrawCStrVerb, `TextPtr` points to the text buffer or C string to draw. |
| `TextLength` | $D8 | If the draw verb is DrawTextVerb, `TextLength` contains the number of bytes in the text buffer. |
| `CharToDraw` | $D6 | If the draw verb is DrawCharVerb, `CharToDraw` contains the character to draw. |

**StdLine**

| | | |
|---|---|---|
| `Y1` | $A6 | Starting Y value for the line to draw |
| `X1` | $A8 | Starting X value for the line to draw |
| `Y2` | $AA | Ending Y value for the line to draw |
| `X2` | $AB | Ending X value for the line to draw |
| `Rect2` | $AE | Exactly the same thing as `Y1`, `X1`, `Y2` and `X2` in the top, left, bottom, and right of the rectangle |

**StdRect**

| | | |
|---|---|---|
| `DrawVerb` | $38 | One of the following five drawing verbs:<br>Frame 0<br>Paint 1<br>Erase 2<br>Invert 3<br>Fill 4 |
| `Rect1` | $A6 | The rectangle to draw in standard form (top, left, bottom, right) |

| | | |
|---|---|---|
| `TheFillPat` | $10E | The pattern to use for the rectangle if the verb is Fill |

**Note:** The QuickDraw II Auxiliary `SpecialRect` call does not use the rectangle bottleneck procedures.

**StdRRect**

| | | |
|---|---|---|
| DrawVerb | $38 | One of the following five drawing verbs:<br>Frame 0<br>Paint 1<br>Erase 2<br>Invert 3<br>Fill 4 |
| Rect1 | $A6 | The boundary rectangle for the round rectangle |
| OvalRect | $295 | A copy of the boundary rectangle for the round rectangle |
| OvalHeight | $208 | The oval height for the rounded part of the round rectangle |
| OvalWidth | $20A | The oval width for the rounded part of the round rectangle |
| • ArcAngle | $D2 | Must be 360 |
| • StartAngle | $D4 | Must be zero |
| TheFillPat | $10E | The pattern to use for the round rectangle if the verb is Fill |

**StdOval**

| | | |
|---|---|---|
| DrawVerb | $38 | One of the following five drawing verbs:<br>Frame 0<br>Paint 1<br>Erase 2<br>Invert 3<br>Fill 4 |
| Rect1 | $A6 | The boundary rectangle for the oval |
| OvalRect | $295 | A copy of the boundary rectangle for the oval |
| • OvalHeight | $208 | Must be the height of the oval |
| • OvalWidth | $20A | Must be the width of the oval |
| • ArcAngle | $D2 | Must be 360 |
| • StartAngle | $D4 | Must be zero |
| TheFillPat | $10E | The pattern to use for the oval if the verb is Fill |

**StdArc**

| | | |
|---|---|---|
| DrawVerb | $38 | One of the following five drawing verbs:<br>Frame 0<br>Paint 1<br>Erase 2<br>Invert 3<br>Fill 4 |
| Rect1 | $A6 | The boundary rectangle for the arc |
| • OvalWidth | $20A | Must be the width of the boundary rectangle for the arc |
| ArcAngle | $D2 | The number of degrees the arc will sweep |
| StartAngle | $D4 | The starting position of the arc |
| TheFillPat | $10E | The pattern to use for the arc if the verb is Fill |

**StdPoly**

| | | |
|---|---|---|
| `DrawVerb` | $38 | One of the following five drawing verbs:<br>Frame 0<br>Paint 1<br>Erase 2<br>Invert 3<br>Fill 4 |
| `RgnHandleA` | $50 | The handle to the polygon data structure |
| `TheFillPat` | $10E | The pattern to use for the polygon if the verb is Fill |

**StdRgn**

| | | |
|---|---|---|
| `DrawVerb` | $38 | One of the following five drawing verbs:<br>Frame 0<br>Paint 1<br>Erase 2<br>Invert 3<br>Fill 4 |
| `RgnHandleC` | $70 | The handle to the region to draw |
| `TheFillPat` | $10E | The pattern to use for the region if the verb is Fill |

**StdPixels**

| | | |
|---|---|---|
| `SrcLocInfo` | $CC | The `LocInfo` record for the source pixel map |
| `DestLocInfo` | $0C | The `LocInfo` record for the destination pixel map |
| `SrcRect` | $DC | The source rectangle for the operation in local coordinates for the source pixel map (as described in the source `LocInfo` record) |
| `DestRect` | $1C | The destination rectangle for the operation in local coordinates for the destination pixel map (as described in the destination `LocInfo` record) |
| `XferMode` | $E4 | The mode to use for data transfer |
| `RgnHandleA` | $50 | The handle to the first region to which drawing is clipped (usually the `ClipRgn` from the `GrafPort`) A `NIL` handle is not allowed. To signify no clipping, pass a handle to the `WideOpen` region, which is defined as 10 bytes:<br>`Length` $A (word)<br>`-MaxInt` -$3FFF (word)<br>`-MaxInt` -$3FFF (word)<br>`+MaxInt` +$3FFF (word)<br>`+MaxInt` +$3FFF (word) |
| `RgnHandleB` | $60 | The handle to the second region to which drawing is clipped (usually the `VisRgn` from the `GrafPort`) A `NIL` handle is not allowed. To signify no clipping, pass a handle to the `WideOpen` region. |

| | | |
|---|---|---|
| `RgnHandleC` | $70 | The handle to the second region to which drawing is clipped (usually the mask region from the `CopyPixels` or the `PaintPixels` call) A `NIL` handle is not allowed. To signify no clipping, pass a handle to the `WideOpen` region. |

**StdComment**

| | | |
|---|---|---|
| TheKind | $A6 | The kind of input for the comment |
| TheSize | $A8 | The number of bytes to put into the picture |
| TheHandle | $AA | The data to put into the picture |

**StdTxMeas**

| | | |
|---|---|---|
| DrawVerb | $38 | Describes the kind of text to draw. There are three possible values:<br>DrawCharVerb 0<br>DrawTextVerb 1<br>DrawCStrVerb 2 |
| TextPtr | $DA | If the draw verb is DrawTextVerb or DrawCStrVerb, TextPtr points to the text buffer or C string to draw. |
| TextLength | $D8 | If the draw verb is DrawTextVerb, TextLength contains the number of bytes in the text buffer. |
| CharToDraw | $D6 | If the draw verb is DrawCharVerb, CharToDraw contains the character to measure. |
| TheWidth | $DE | The resulting width should be put here. |

**StdTxBnds**

| | | |
|---|---|---|
| DrawVerb | $38 | Describes the kind of text to draw. There are three possible values:<br>DrawCharVerb 0<br>DrawTextVerb 1<br>DrawCStrVerb 2 |
| TextPtr | $DA | If the draw verb is DrawTextVerb or DrawCStrVerb, TextPtr points to the text buffer or C string to draw. |
| TextLength | $D8 | If the draw verb is DrawTextVerb, TextLength contains the number of bytes in the text buffer. |
| CharToDraw | $D6 | If the draw verb is DrawCharVerb, CharToDraw contains the character to draw. |
| RectPtr | $D2 | Indicates the address to put the resulting rectangle. |

**StdGetPic**

This call takes input on the stack rather than the direct page. This is the one standard bottleneck procedure which you call with the direct page register set to something other than the direct page of QuickDraw II; it is set to a part of the stack.

Stack Diagram on Entrance to StdGetPic

| | |
|---|---|
| Previous Contents | |
| DataPtr | Pointer to destination buffer |
| Count | Integer (unsigned) (bytes to read) |
| RTL Address | 3 bytes |

```
----------------    Top of Stack
```

Stack Diagram just before exit from `StdGetPic`

```
Previous Contents
RTL Address         3 bytes
----------------    Top of Stack
```

**StdPutPic**

This call takes input on the stack rather than the direct page; however, unlike `StdGetPic`, the direct page for QuickDraw II is active when you call this routine.

Stack Diagram on Entrance to `StdPutPic`

```
Previous Contents
DataPtr             Pointer to source buffer
Count               Integer (unsigned) (bytes to read)
RTL Address         3 bytes
----------------    Top of Stack
```

Stack Diagram just before exit from `StdPutPic`

```
Previous Contents
RTL Address         3 bytes
----------------    Top of Stack
```

## Dealing with the Cursor

The cursor can get in your way when you want to draw directly to the screen. QuickDraw II has two low-level routines which help you avoid this problem: `ShieldCursor` and `UnshieldCursor`. `ShieldCursor` tells QuickDraw II to hide the cursor if it intersects the `MinRect` and to prevent the cursor from moving until you call `UnshieldCursor`.

There is a bug in `ShieldCursor` for System Disks 4.0 and earlier. This bug is related to the routine `ObscureCursor`. When the cursor is obscured, `ShieldCursor` does not prevent the cursor from moving; therefore, the user is able to move the cursor during a QuickDraw II operation, and this movement may disturb the screen image.

Calls to `ShieldCursor` **must** be balanced by calls to `UnshieldCursor`. You may not call `ShieldCursor` successively without calling `UnshieldCursor` after each call to `ShieldCursor`. There is no error checking, so careless use of these routines will result in an unusable system.

`MinRect` is the smallest possible rectangle which encloses all the pixels that may be affected by a drawing call. You keep `MinRect` on the direct page and usually calculate it by intersecting the rectangle of the object you are drawing with the `BoundsRect`, `PortRect`, boundary box of the `VisRgn`, and the boundary box of the `ClipRgn`. You must set up `MinRect` yourself.

ShieldCursor also looks at two other fields on the direct page of QuickDraw II. ImageRef is a long word located at $0E. If ImageRef does not point to $E12000 or $012000, QuickDraw II assumes you are not drawing to the screen, so it does not have to shield the cursor. BoundsRect is a rectangle located at $14, and QuickDraw II uses it to translate MinRect into global coordinates. These values are generally correct, but under the following known circumstance, they are not and ShieldCursor does not function properly:

1. You have just drawn to an off-screen GrafPort with QuickDraw II.
2. You switch to a GrafPort on the screen.
3. You call ShieldCursor.

ImageRef and BoundsRect are not updated until QuickDraw II is actually committed to drawing, thus, these values are still for the off-screen GrafPort in this case, even though you switched to a GrafPort on the screen. Therefore, when you call ShieldCursor, you have to make sure that these values are current. (If these values are current, ShieldCursor will work correctly, no matter what the circumstances.)

You can find the location of the QuickDraw II direct page with the GetWAP call. For speed reasons, you may not want to make the GetWAP call for each ShieldCursor call. You may wish to get the work area pointer value after starting QuickDraw II and store it for future reference.

Calling ShieldCursor:

1. Set direct page for QuickDraw II.
2. Save the existing values of MinRect, ImageRef, and BoundsRect.
3. Set MinRect, ImageRef, and BoundsRect.
4. Let QuickDraw II know you've changed the contents of its direct page by clearing the "dirty" flags bits 14 to 0:

```
DirtyFlags    equ    $EC

              ldx    #$200         ;index to QD's third page of work space
              lda    DirtyFlags,x
              and    #$8000
              sta    DirtyFlags,x
```

5. JSL to ShieldCursor.
6. Restore the previous values of MinRect, ImageRef, and BoundsRect.

**Note**: Saving and restoring these values was not previously mentioned in this Note and in most circumstances it is not necessary. Saving and restoring is now recommended. In particular, if ShieldCursor is called inside a QuickDraw II bottleneck procedure, the system can crash if you fail to restore the contents of direct page.

Calling UnshieldCursor:

1. Set direct page for QuickDraw II.

2. JSL to UnshieldCursor.

| | |
|---|---|
| **ShieldCursor** | **$E01E98** |
| MinRect | $00 |
| ImageRef | $0E |
| BoundsRect | $14 |
| **UnshieldCursor** | **$E01E9C** |

## Further Reference

- *Apple IIGS Toolbox Reference*, Volume 2



# Apple IIGS
# #35: Printer Driver Specifications

Revised by: Matt Deatherage September 1990
Written by: Dan Hitchens, Matt Deatherage & Suki Lee May 1988

This Technical Note describes the routines and internal structures needed to design a printer driver for the Apple IIGS system, and you should use this Note with the Apple IIGS Toolbox Reference manuals. An overview and associated parameters for each of the printer driver routines are in the Print Manager chapter, and you should refer to these for a complete picture.
**Changed since March 1990:** Added corrections and further descriptions.

## Printing Modes

There are two printing modes: immediate and deferred.

- In **immediate mode**, pages are printed as they are drawn into the printing `grafPort`. As the application makes QuickDraw II calls, the printer driver immediately generates commands, transferring ink to page when the page is closed. This is the fastest form of printing, but only produces high-quality images on printers that can translate QuickDraw II commands to other graphic commands. For example, the LaserWriter driver translates the QuickDraw II calls into PostScript® calls which can produce high-quality images.

- In **deferred mode** (sometimes referred to as **spool mode**), pages are captured to memory or disk and printed after all pages have been defined. Most printer drivers use deferred mode to create high-quality images. Since most drivers cannot obtain enough memory to image an entire page at once, they redraw page in several pieces, or **bands**. The printer driver creates a `grafPort` whose `boundsRect`, `portRect`, `clipRgn`, and `visRgn` correspond to the band and plays the picture back, thus causing the saved commands to draw only the images which fall within the band. Once the pixel image for the band is created, the printer driver converts the image to printer codes and sends the codes to the printer through the port driver.

## File Structure

The user can install new printer drivers into the system by copying a printer driver file into a subdirectory called DRIVERS within the SYSTEM subdirectory. The printer driver file must be of type $BB and have an auxiliary type of $0001.

## Print Driver Calls

A printer driver must support the following calls:

| | | |
|---|---|---|
| `PrDefault` | $0913 | Sets print record to default |
| `PrValidate` | $0A13 | Validates print record |
| `PrStlDialog` | $0B13 | Performs a style dialog |
| `PrJobDialog` | $0C13 | Performs a job dialog |
| `PrPixelMap` | $0D13 | Prints a pixel map |
| `PrOpenDoc` | $0E13 | Opens the document |
| `PrCloseDoc` | $0F13 | Closes the document |
| `PrOpenPage` | $1013 | Opens a page |
| `PrClosePage` | $1113 | Closes a page |
| `PrPicFile` | $1213 | Prints a picture file |
| --RESERVED-- | $1313 | |
| `PrError` | $1413 | Gets the error value |
| `PrSetError` | $1513 | Sets the error value |
| `GetDeviceName` | $1713 | Gets device's name |
| `PrDriverVer` | $2313 | Gets installed driver version |

Printer drivers **may** support the following calls if they use the new driver structure outlined below:

| | | |
|---|---|---|
| `PrGetPrinterSpecs` | $1813 | Returns printer type and characteristics |
| `PrGetPgOrientation` | $3813 | Returns page orientation |

## Print Driver Entry

- For older drivers, entry is at the first byte (no offset). For newer (Print Manager 3.0 and later) drivers, the first word is $0000, indicating a new style driver. The next word is a count of how many calls this driver supports. All drivers **must** support the minimum call set. Additional calls must be supported in the sequence listed (for example, if a driver supports `PrGetPgOrientation`, it must also support `PrGetPrinterSpecs`).
- The Print Manager places an index to the correct routine in the `X` register (see the example and note the specific ordering of the routines) .
- There are two long return addresses (six bytes) that have been pushed onto the stack. (You must take these addresses into account to access the parameters and to return correctly.)

### Example

```
StartOfNewDriver    START

                    dc i2 '0'                          ; new style driver
                    dc i2'(ListEnd-PrDriverList)/4'    ; count

                    jmp (PrDriverList,x)
```

```
PrDriverList        dc a4'PrDefault'
                    dc a4'PrValidate'
                    dc a4'PrStlDialog'
                    dc a4'PrJobDialog'
                    dc a4'PrDriverVer'
                    dc a4'PrOpenDoc'
                    dc a4'PrCloseDoc'
                    dc a4'PrOpenPage'
                    dc a4'PrClosePage'
                    dc a4'PrPicFile'
                    dc a4'InvalidRoutine'
                    dc a4'PrError'
                    dc a4'PrSetError'
                    dc a4'GetDeviceName'
                    dc a4'PrPixelMap'
                    dc a4'PrGetPrinterSpecs'
                    dc a4'PrGetPgOrientation'
ListEnd             anop
```

In previous versions of this Note, the `PrPixelMap` and `PrDriverVer` entries were reversed.

Note that when using the above technique, you're using a 16-bit jump into a table of 24-bit addresses. If all your entry points are in the same segment, this is not a problem.

If your routines' entry points are not all in the same segment, you need a dispatching routine like the following:

```
StartOfNewDriver    START

                    dc i2 '0'                           ; new style driver
                    dc i2 '(ListEnd-PrDriverList)/4'    ; count

                    lda PrDriverList+2,x
                    sep #$20
                    pha                                 ; push high byte of address
                    rep #$20
                    lda PrDriverList,x
                    dec a                               ; decrement low 2 bytes only
                    pha                                 ; push modified low word of
address
                    rtl                                 ; transfer to the routine
```

See Apple IIGS Technical Note #90, 65816 Tips and Pitfalls, for a discussion of dispatching with `RTL`.

## Print Driver Exit

When one of your routines is ready to exit, it needs to remove the input parameters from the stack, leaving the result space (if any) and the two `RTL` addresses. Set the accumulator and the carry flag to reflect any error you are returning, then perform an `RTL`.

**Example**

If there are N bytes of input parameters to remove, use something like the following. This code assumes that the error code is in the accumulator.

```
                    tay                                 ; keep error code in Y
temporarily
                    lda 5,s
                    sta N+5,s
                    lda 3,s
                    sta N+3,s
                    lda 1,s
                    sta N+1,s
                    tsc
                    clc
                    adc #N
                    tcs
                    tya                                 ; get error code
                    cmp #1                              ; set carry if error is not
zero
                    rtl
```

Figure 1 diagrams the stack just before exiting the print driver:

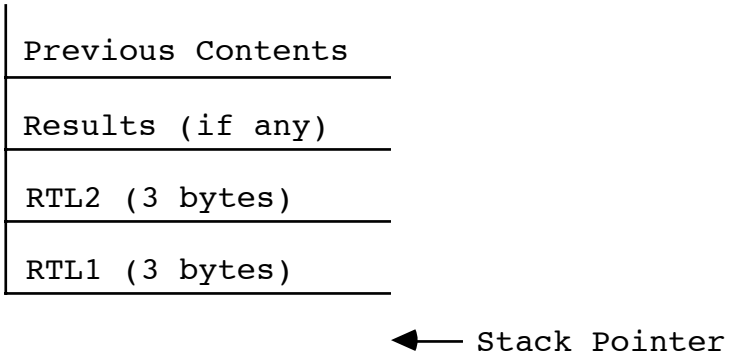



**Figure 1–Stack Prior to Exiting the Print Driver**

You should do an `RTL` with the contents of the flags and registers set appropriately. (See the Return from Call section of the "Using The Apple Tools" chapter of the *Apple IIGS Toolbox Reference*.)

## Print Record Structure

Since application programs often need to fiddle with parts of the print record (i.e., the values in the style subrecord), we have defined ways for applications to interpret the print record, and specifically the style subrecord.

`iDev`, the first word of the printer information subrecord, has two defined values for third-party printer drivers. A value of $8001 indicates a dot-matrix printer while a value of $8003 indicates a laser printer.

A value of $8001 indicates that fields of the style subrecord should be interpreted as they are by the ImageWriter driver, as documented in the *Apple IIGS Toolbox Reference*. The first seven bits (0–6) of `wDev` are defined as for the ImageWriter driver. Bits 7-11 are reserved for Apple's use and must be set to zero. Bits 12-15 may be used by third-party printer drivers as necessary; these bits are set to zero in Apple's drivers.

A value of $8003 indicates that fields of the style subrecord should be interpreted as they are by the LaserWriter driver. The first four bits (0–3) of `wDev` are defined as for the LaserWriter driver. Bits 4-11 are reserved for Apple's use and must be set to zero. Bits 12-15 may be used by third-party printer drivers as necessary; these bits are set to zero in Apple's drivers.

If an application wishes to take advantages of specific features of a third-party printer driver, it has to know that it is dealing with that driver. Since all drivers look pretty much alike, the Print Manager allows you to ask for the name of the currently selected printer driver. An application may make the Print Manager call `PMGetPrinterName`, which is documented in Volume 3 of the *Toolbox Reference*. The Print Manager returns the name of the currently selected printer in a Pascal (length byte) string. The name returned is the name of the **file** from which the driver was loaded. If you intend to use this method to identify a driver, you must inform users **not** to rename the Printer Driver file on the boot disk.

For alternate driver identification, Developer Technical Support assigns new `iDev` values if you feel it is absolutely necessary for your driver. Please keep in mind, however, that no application knows how to interpret style records for non-standard `iDev` values, and that Apple does not publish such interpretations.

## Print Driver Calls

Your printer driver handles the following calls:

**PrDefault ($0913)**

Description:
Fills the fields of the specified print record with default values for the printer.

Passed:

| | | |
|---|---|---|
| `PrintRecordHandle` | LONG | Handle to the print record |

Returned:
None

Performs the following:
- Validates that `PrintRecordHandle` is a handle and does nothing if not.
- Determines the default values for the print record either through tables or calculations. The default values should take into account such things as paper size and orientation, print mode, printer type, etc.

- Copies the default values to the print record specified by the `PrintRecordHandle` parameter.

**PrValidate ($0A13)**

Description:
Checks the print record to see that it is valid for the currently installed printer driver.

Passed:

| | | |
|---|---|---|
| `PrintRecordHandle` | LONG | Handle to the print record |

Returned:

| | | |
|---|---|---|
| `ChangeFlag` | WORD | Boolean; TRUE if the record is adjusted |

Performs the following:
- Checks to see if the print record is from this particular driver.
- If the print record is not from this driver, it uses the default values for this driver.
- If the print record is from this driver, it makes any changes that might be needed (i.e., style, paper size, etc.).

**PrStlDialog ($0B13)**

Description:
Performs a style dialog with the user.

Passed:

| | | |
|---|---|---|
| `PrintRecordHandle` | LONG | Handle to the print record |

Returned:

| | | |
|---|---|---|
| `ConfirmFlag` | WORD | Boolean; TRUE if the dialog is confirmed |

Performs the following:
- Conducts a style dialog with the user to determine the page dimensions and other information needed for page setup (the initial settings of the dialog are derived from the print record).
- If the user confirms the dialog, the information from the dialog is saved in the specified print record, `PrValidate` is called, and the routine returns TRUE.
- If the user cancels the dialog, the print record is left unchanged, and the routine returns FALSE.

**Note:** The following are items typically found in printer style dialogs:

- Paper Size (US Letter, US Legal, A4 Letter, B5 Letter, International Fanfold)
- Printing Orientation (Landscape, Portrait)
- Vertical Sizing (Normal, Intermediate, Condensed)
- Special Effects:
  - Font Effects (Font Substitution, Smoothing)
  - Reduction or Enlargement
  - Gaps or No Gaps between pages

Every printer style dialog should have an OK button (default) and a Cancel button.

**Note:** When calling other routines in your printer driver (like `PrValidate`), be sure to do so through the Tool Dispatcher ($E10000 or $E10004) so any necessary patches have an opportunity to execute.

**PrJobDialog ($0C13)**

Description:
Performs a job dialog with the user.

Passed:

| | | |
|---|---|---|
| `PrintRecordHandle` | LONG | Handle to the print record |

Returned:

| | | |
|---|---|---|
| `ConfirmFlag` | WORD | Boolean; True if the dialog is confirmed |

Performs the following:
- Conducts a job dialog with the user to determine the print quality, range of pages to print, and other specifications. The initial settings are derived from the previous `PrJobDialog` call (or initial default values) except the page range which is set to ALL, and the number of copies which is set to ONE.
- If the user confirms the dialog, `PrValidate` is called, the print record is updated, and the routine returns TRUE.
- If the user cancels the dialog, the print record is left unchanged, and the routine returns FALSE.

**Note:** The following are items typically found in printer job dialogs:

- Print Quality (Best, Faster, Draft, etc.)
- Color option
- Pages (All, Range)
- Copies
- Paper Source (paper cassette, manual feed)

Every printer job dialog should have an OK button (default) and a Cancel button.

**Note:** When calling other routines in your printer driver (like `PrValidate`), be sure to do so through the Tool Dispatcher ($E10000 or $E10004) so any necessary patches have an opportunity to execute.

**PrPixelMap ($0D13)**

Description:
Prints all or part of the specified pixel map.

Passed:

| | | |
|---|---|---|
| `srcLocPtr` | LONG | Pointer to the source `LocInfo` which contains the pointer to the pixel map. |
| `srcRectPtr` | LONG | Pointer to the rectangle which encloses the pixel map to be printed. |
| `colorFlag` | WORD | Boolean; FALSE if black and white, TRUE if color. |

Returned:
None
Performs the following:
- Calls `DevIsItSafe` (port driver call) to verify that the port it functioning and it is safe to proceed. If it is not functioning, set the internal error code to $1302 (`Port Not On`) and return with an error status.
- Saves the current `grafPort`.
- Turns on the watch cursor to signal the user that it will take some time.
- Clears the internal error code (default, if no errors occur).

You can choose to print the pixel map in any convenient fashion; one convenient way is to allocate a new print record and call your normal printing routines. This method is outlined below.

- Gets a new handle for a print record and set it to the defaults by calling `PrDefault`.
- If `colorFlag` is set, change the style subrecord of the print record to reflect color printing.
- Do any initialization that might be needed by the driver.
- Determine the intersection of the two rectangles (rectangle pointed to by `srcRectPtr` and the pixel map's boundary rectangle from `srcLocPtr`) and if there is no intersection, then nothing is to be printed.
- Print the pixel image which is within the intersection of the two rectangles.
- Cause a page eject to occur on the printer.
- Do any clean up that is needed.
- Turn off the watch cursor by calling `InitCursor` (or restore the previous cursor using `SetCursor`).
- Restore the `grafPort` by calling `SetPort`.

**PrOpenDoc ($0E13)**

Description:
This routine initializes the things needed to open a document. In deferred mode, it establishes a `grafPort` and makes it the current port for printing.

Passed:

| | | |
|---|---|---|
| `PrintRecordHandle` | LONG | Handle to the print record |
| `PrinterPortPtr` | LONG | Pointer to the `grafPort`, if desired, zero to allocate a new `grafPort` |

Returned:

| | | |
|---|---|---|
| `PrinterPortPtrRet` | LONG | Pointer to the `grafPort` if the `PrinterPortPtr` was zero |

Performs the following:
- Calls `DevIsItSafe` (port driver call) to verify that the port is functioning and it is safe to proceed.
- Turns on the watch cursor to signal the user that it will take some time.
- Validates the print record passed by calling `PrValidate`.

- Clears the internal error code (default, if nothing happens).
- Puts up a dialog indicating that printing is occurring (or preparing to print).
- If the user needs a `grafPort`, create one and internally note that one was created (`PrCloseDoc` needs to know that one was created here).
- Initializes parameters (i.e., page number, document number, etc.).
- If deferred mode, create an initial page list (an array of handles to pictures) for recording pages. You can pick an arbitrary number to start with (like 20). This assumes spooling to memory; spooling to disk will obviously be different.
- Do other initialization that might be needed to start a print job.

Possible errors:

| | | |
|---|---|---|
| `portNotOn` | $1302 | Indicates Port Not On |

**PrCloseDoc ($0F13)**

Description:

Closes the `grafPort` being used for printing. For immediate mode, this routine ends the printing job. For deferred mode, this routine ends the recording of the document to be printed.

Passed:

| | | |
|---|---|---|
| `PrintGrafPortPtr` | LONG | Pointer to the `grafPort` used for printing |

Returned:

None

Performs the following:

- Checks that the last print driver call did not cause a Port Not On error. If the error occurred, do nothing and return.
- Call `ClosePort` to close the printing `grafPort`.
- If the driver allocated a `grafPort` in `PrOpenDoc`, disposes of it.
- If in immediate mode, does what is needed to shut things down.
- Takes down the information dialog box from `PrOpenDoc`.

Possible errors:

| | | |
|---|---|---|
| `portNotOn` | $1302 | Indicates Port Not On |
| `prBozo` | $13FF | Someone unloaded the driver in the middle of the print loop |

**PrOpenPage ($1013)**

Description:

Begins a new page only if the page falls within the page range specified in the job subrecord.

Passed:

| | | |
|---|---|---|
| `PrintGrafPortPtr` | LONG | Pointer to the `grafPort` used for printing |
| `PageFramePtr` | LONG | Pointer to the `scaling` parameter, zero for none. |

Returned:

None

Performs the following:

- Looks at the driver's internal error value, and if an error has occurred, it returns without doing anything.
- Increments the page number.
- Calls `SetPort` to make the specified port the current port.
- Initializes the port and zeroes the boundary rectangle so no actual drawing occurs.
- If immediate mode, then do the following:

  If this page is to be printed, install immediate mode procedures by doing the following:

- Create a procedure table (get the standard procedures from `SetStdProcs`).
- Put pointers to your procedures into the table and call the QuickDraw II routine `SetGrafProcs`. This causes QuickDraw II calls to call your routines instead of drawing to the pixel map associated with the `grafPort`.

- If deferred mode, then do the following:
    - If the current page is out of the page range, then return without doing anything further.
    - If the user passes his own `PageFramePtr` , then get it.
    - Open a picture by calling `OpenPicture` and adding its handle to the page list array described in `PrOpenDoc`.
    - Set the `ClipRgn` and `VisRgn` to the sizing framing rectangle specified by PageFramePtr , or if none was specified, to the default of `rPage`.

Possible errors:

| | | |
|---|---|---|
| `portNotOn` | $1302 | Indicates Port Not On |
| `prBozo` | $13FF | Someone unloaded the driver in the middle of the print loop |

**PrClosePage ($1113)**

Description:
This signals the end of a page.

Passed:

| | | |
|---|---|---|
| `PrintGrafPortPtr` | LONG | Pointer to the `grafPort` used for printing |

Returned:
None

Performs the following:
- Looks at the driver's internal error value and if a Port Not On error has occurred, it returns without doing anything.
- If immediate mode, do the following:
    - If the current page is within the range of pages to be printed, then cause a form feed (unless no gap was specified).
- If deferred mode, do the following:
    - If there was no picture generated, then do nothing and just return.
    - Restore the `grafPort` to the port saved in `PrOpenPage`.
    - Do a `ClosePicture` to close the picture.

Possible errors:

| | | |
|---|---|---|
| `portNotOn` | $1302 | Indicates Port Not On |
| `prBozo` | $13FF | Someone unloaded the driver in the middle of the print loop |

**PrPicFile ($1213)**

Description:
Prints a picture file generated in deferred mode.

Passed:

| | | |
|---|---|---|
| `PrintRecordHandle` | LONG | Handle to the print record |
| `PrintGrafPortPtr` | LONG | Pointer to the `grafPort` used for printing |
| `StatusRecPtr` | LONG | Pointer to the printer status record |

Returned:
None

Performs the following:
- Looks at the driver's internal error value and if a Port Not On error has occurred, it returns without doing anything.
- If immediate mode, return without doing anything.
- If deferred mode, then do the following:
  - If the error code is not zero (errors) then dispose of all the recorded page images.
  - Put up an information dialog indicating that printing is occurring.
  - Display a watch cursor (saving the current cursor first if you like).
  - If `PrintGrafPortPtr` is `NIL`, create one and make a note of it.
  - Call `OpenPort` to make the `grafPort` the current port.
  - If `StatusRecPtr` is `NIL`, use an internal one. This is to simplify your code; if the `StatusRecPtr` is `NIL`, you can reasonably choose not to use a status record at all, but this requires an extra code path.
  - Initialize the status record and the number of copies counter.
  - If the idle procedure pointer in the print record is `NIL`, point to an internal one. Again, as with the `StatusRecPtr`, you can choose to ignore idle procedures if no pointer is provided, but this requires an extra code path.
  - **Do The Following For Each Copy**:
    - Calculate the number of bands to print one page and initialize the page counter.
    - **Do The Following For Each Page:**
      - Call the idle procedure routine and initialize the band counter.
      - Get the handle to the picture associated with the current page.
      - Set the dirty flag in the status record to FALSE.
      - If manual paper feed, put up a dialog and wait for a response.
      - **Do The Following For Each Band:**
        - Call the idle procedure.
        - Calculate the band rectangle and update `iCurBand` with the current band number.
        - Call the idle procedure again.
        - Set the imaging flag in the status record to TRUE.
        - Call `InitPort` to reinitialize the port.
        - Adjust fields in the port to cause drawing into the band buffer.
        - Adjust fields in the location information field of the status record and calculate the sizing rectangle.
        - Calculate the boundary rectangle for the band and set the port rectangle to it.
        - Set the `ClipRgn` and the `VisRgn` to the sizing rectangle.
        - Initialize the band by filling it with white space.
        - Call `DrawPicture` to draw the picture into the band's rectangle.
        - Do whatever is needed to print the pixel image in the band's rectangle.
        - Clear the imaging flag.

        - Calculate the next band's position.
        - Increment the band's counter and loop back if not done.
    - If a vertical gap was specified, cause a form feed.
    - Increment the page count to the next page and loop back if not done.
  - Increment the number of copies counter and loop back if not done.
- Free any buffers that you own and close the port.
- Dispose of the information dialog that you put up.
- Dispose of each picture in the picture list by calling `KillPicture`.
- Dispose of the picture list itself.
- Restore the cursor.

Possible errors:

| | | |
|---|---|---|
| `portNotOn` | $1302 | Indicates Port Not On |
| `prBozo` | $13FF | Someone unloaded the driver in the middle of the print loop |

**PrError ($1413)**

Description:
Gets the error code from the last Print Manager call.

Passed:
None

Returned:

| | | |
|---|---|---|
| `LastError` | WORD | Result code from last Print Manager call |

Performs the following:
- Gets the driver's internal error value (which was determined by the last driver call) and sets the return parameter `LastError` to it.

Possible Errors:

| | | |
|---|---|---|
| `noError` | $0000 | |
| `PrAbort` | $0080 | Indicates print job was aborted<br>$1301 Indicates missing drivers<br>$1302 Indicates Port Not On<br>$1303 Indicates No Print Record<br>$1306 Indicates PAP Connection Not Made<br>$1307 Indicates Read/Write PAP Error<br>$1308 Indicates Printer Connection Failed |
| `prBozo` | $13FF | Someone unloaded the driver in the middle of the print loop |

**PrSetError ($1513)**

Description:
Sets the error value.

Passed:

| | | |
|---|---|---|
| `ErrorNumber` | WORD | Error number to be set |

Returned:
None

Performs the following:
- Sets the driver's internal error value to the value of the passed `ErrorNumber` parameter.

**GetDeviceName ($1713)**
(also known as **PrChanged**)

Description:
Used as a communications tool between the printer driver and port driver.

Passed:
None

Returned:
None

Performs the following:

- Calls the port driver routine `PrDevPrChanged` with the printer name as input. This is necessary for drivers that work over AppleTalk. The name passed as the parameter to `PrDevPrChanged` should be what AppleTalk uses in an `NBPLookup` situation; for AppleTalk, such a name should follow Name Binding Protocol conventions.

  This routine will be called by the Print Manager when your driver is first loaded so a network port driver can find devices of your type. Applications should not make this call. When this routine will be called is not guaranteed; you can't use this as a substitute for a startup call.

**PrDriverVer ($2313)**

Description:
Returns the version number of the currently installed printer driver.

Passed:

| | | |
|---|---|---|
| `Wordspace` | WORD | Space for results |

Returned:

| | | |
|---|---|---|
| `versionInfo` | WORD | Printer driver's version number |

Performs the following:

- Gets the internal version number of the printer driver and returns it on the stack at `versionInfo`.

**Note:** The internal version number is stored major byte, minor byte (i.e., $0103 represents version 1.3)

**PrGetPrinterSpecs ($1813)**

Description:
Returns the type of printer and the printer's characteristics.

Passed:

| | | |
|---|---|---|
| `Wordspace` | WORD | Space for results |

| | | |
|---|---|---|
| `Wordspace` | WORD | Space for results |

Returned:

| | | |
|---|---|---|
| `PrinterType` | WORD | 0 = undefined<br>1 = ImageWriter or ImageWriter II<br>2 = ImageWriter LQ<br>3 = LaserWriter family (except IISC)<br>4 = Epson<br>$8001 = generic dot matrix printer<br>$8003 = generic laser printer |
| `PrCharacteristics` | WORD | Bits 15 - 2 = reserved, must be zero<br>Bits 1-0: 00 = cannot determine<br>01 = black and white only<br>10 = color capable |

Performs the following:

- Returns characteristics intrinsic for the printer being supported.

The value returned for `PrinterType` should be the driver's `iDev` value.

**PrGetPgOrientation ($3813)**

Description:

Returns the page orientation from a print record.

Passed:

| | | |
|---|---|---|
| `Wordspace` | WORD | Space for result |
| `PrintRecordHandle` | LONG | Handle to the print record |

Returned:

| | | |
|---|---|---|
| `PgOrientation` | WORD | Current page orientation:<br>0 = portrait<br>1 = landscape |

Performs the following:

- Returns the page orientation from the current page setup information in the print record.

## Immediate Mode Procedures

To print in the immediate mode, you need to install procedures which cause printing when you make QuickDraw II calls (as noted in `PrOpenPage`). This section describes the structure and parameters for these routines.

The basic idea is that your driver replaces low-level QuickDraw II routines with pointers to your own routines. For example, when someone wants QuickDraw II to draw some text (say with `DrawString`), QuickDraw II calls your low-level routine to draw the text. You can then print the text instead.

To install the immediate mode procedures, first create a procedure table for sixteen entries (16*4 bytes) and fill it with the standard procedures by calling `SetStdProcs`. Once you have the standard procedures, install the addresses of your replacement procedures into it and call `SetGrafProcs`. Installing your procedure addresses causes the appropriate QuickDraw II calls to call your procedures, which, in turn, perform the actual printing.

The routines that need to be written are known as QuickDraw II "bottleneck procedures." For most dot-matrix printer drivers, the one of most concern when writing immediate mode procedures is `StdText`. If your target device has an alternate page imaging language, you may wish to print entirely in immediate mode. In this case, you want to intercept most of the bottleneck procedures. Apple IIGS Technical Note #34, Low-Level QuickDraw II Routines, contains information on how to install these procedures. The sample code which follows shows how to replace `StdPixels` and `StdText`.

**Example:**

```
;****************************************************************
;** Example of Immediate Mode Printer Procedures.              **
;****************************************************************
Immedprocs     Start

SrcRect        equ $DC
SrcLocInfo     equ $CC
DrawVerb       equ $38
TextPtr        equ $da
TextLength     equ $d8
CharToDraw     equ $d6

;----------------------------------------------------------------
;
; StdPixels Procedure (Prints Pixel maps)
;
;----------------------------------------------------------------
Pixel          Entry

               phb                          ;save data bank reg on stack
               phk                          ;get program bank reg.
               plb                          ;use as data bank reg.

               lda iPrErr                   ;get errors
               beq Continue                 ;branch if none
               brl ExitPixel                ;branch if errors

Continue       anop

;This gets the source rectangle and stores it at PixelRect
               ldx #6
MoveSrc        lda SrcRect,x
               sta PixelRect,x
               dex
               dex
               bpl MoveSrc

;This gets the source LocInfo and stores it at PixelLoc
               ldx #16-2
MoveLI         lda SrcLocInfo,x
               sta PixelLoc,x
               dex
               dex
```

```
                bpl MoveLI

                pushlong #PixelLoc          ;push pointer to LocInfo
                pushlong #PixelRect         ;push pointer to rectangle
```

```
;++++++++++++++++++++++
; Insert code here to print a pixel map
;    INPUT:    PixelLoc   LONG, Pointer to pixel LocInfo
;              PixelRect  LONG, Pointer to pixels BoundsRect
;    SP->
;++++++++++++++++++++++

Exitpixel      lda #0                      ;return with no errors
               clc
               plb                         ;restore data bank
               rtl                         ;return with long

PixelLoc       ds 16                       ;pixel LocInfo
PixelRect      ds 8                        ;pixel rectangle

;---------------------------------------------------------------
;
; StdText Procedure (Prints Standard Text)
;
;---------------------------------------------------------------
StdText        Entry

               phb                         ;save data bank reg on stack
               phk                         ;get program bank reg.
               plb                         ;use as data bank reg.

               pushlong #PenPos
               _GetPen                     ;current pen pos. -> PenPos

;++++++++++++++++++++++
; Insert Code Here to move the printers head to the corresponding
; PenPos position (if needed).
;++++++++++++++++++++++

               pushword #0                 ;space for textwidth
                                           ;(for call to _TextWidth)

               lda DrawVerb                ;get DrawVerb
               beq DoCar                   ;if DrawVerb=0 then DoCar

               cmp #1
               beq Dotext2                 ;if DrawVerb=1 then Dotext2
;
;We get here if it's a "C" string (DrawVerb=2)
;
DoCstring      anop
               sep #$20
               longa off
;Search down through string looking for terminator to calc. length
               ldy #0
KeepLooking    lda [TextPtr],y
               beq TheEnd
               iny
               bra KeepLooking
TheEnd         rep #$20
               longa on
               lda TextPtr+2
               pha                         ;push the pointer to string
               lda Textptr
               pha
               phy                         ;push the length of sting
               bra Common
```

```
;
;We get here if it's just one character (DrawVerb=0)
;
DoCar           anop
                pushword #0
                tdc
                clc
                adc #CharToDraw             ;calculate addr. of char.
                pha                         ;push addr. of character
                pushword #1                 ;push length of one char.
                bra Common

;
;We get here if it's a string of text (DrawVerb=1)
;
DoText2         anop
                lda TextPtr+2
                pha                         ;push pointer to the string
                lda Textptr
                pha
                lda TextLength
                pha                         ;push the strings length
Common          lda 5,s                     ;Dup the last 3 words of
                pha                         ;the stack (for _TextWidth)
                lda 5,s
                pha
                lda 5,s
                pha
;++++++++++++++++++++++
; Insert code here to print the text
;
;    INPUT:     TextPointer    LONG, Pointer to text to print
;               TextLength     WORD, No. of bytes to print
;    SP->
;++++++++++++++++++++++
                _TextWidth                  ;get the texts width (DH)
                pushword #0                 ;set (DV)=0
                _Move                       ;move current pen location

ExitText        lda #0                      ;return with no errors
                clc
                plb                         ;restore data bank
                rtl                         ;returnith long

PenPos          ds 4                        ;pen position
                end
```

## Further Reference

- *Apple IIGS Toolbox Reference*, Volumes 1–3
- Apple IIGS Technical Note #36, Port Driver Specifications
- Apple IIGS Technical Note #90, 65816 Tips and Pitfalls