Apple II
Technical Notes

Developer Technical Support

Apple IIGS
#87: Patching the Tool Dispatcher

Written by: Mike Lagae and Dave Lyons
September 1990

This Technical Note presents the Apple standard way to patch into the Apple IIGS Tool Dispatcher vectors.

This Note presents MPW IIGS assembly-language code which provides the Apple-standard way for utilities to patch *and unpatch* the Tool Dispatcher vectors. If **all** Tool Dispatcher patches follow this protocol, patches can be installed and removed in any order, without ever accidentally unpatching somebody who patched in after the one getting removed.

Using this protocol, each patch begins with a header in a standard form—a form recognizable by these routines (see `PatchHeader`). This way routines (like `RemoveE10000`) can scan through the list of patches and remove one from the middle.

If your patch is going to stay in the system until shutdown, use this standard patch protocol anyway. This way other utilities can still recognize your patch and scan past it to find the next one. This Note is not just to show you a way to patch the tool dispatcher—it's to show you **the** way. If you patch tool dispatcher vectors in any **other** way, you strip other utilities of their ability to remove their patches.

Of course, patching the Tool Dispatcher vectors slows down all toolbox calls, so you shouldn't patch the tool dispatcher without a pretty good reason. If you need to patch a toolbox function, it is usually better to do it by modifying a tool set's function pointer table instead of patching the dispatcher.

The code in this note is specific to the System tool dispatch vectors ($E10000 and $E10004), but the same technique is recommended for the User tool dispatch vectors—just change $E10000 to $E10008, $E10004 to $E1000C, and `ToolPointerTable` to `UserToolPointerTable`.

What Is This Stuff?

This Note presents the following four routines.

`PatchHeader` is the simplest patch function that obeys the protocol. This is where you put your own patch code.

`InstallE10000` installs a patch into the patch chain. For example:

```
      pushlong #PatchHeader
      jsl InstallE10000
      ply
      ply                    ;remove the input parameter
      bcc noError            ;error in A
```

`RemoveE10000` removes a patch from the patch chain. For example:

```
      pushlong #PatchHeader
      jsl RemoveE10000
      ply
      ply                    ;remove the input parameter
      bcc noError            ;error in A
```

`CheckPatch` determines whether the specified address is the starting address of a standard patch. For example:

```
      pushlong #PatchHeader
      jsl CheckPatch
      ply
      ply                    ;remove the input parameter
      bcc validPatch
```

First, here are some comments and global equates.

```
***************************************************************************
*
* Patch.e10000 - Routines to patch into the toolbox dispatch
*                vectors at $e10000 and $e10004.
*
* By Michael Lagae
* Software Quality Assurance
* GS Toolbox Test Team
*
* July 18, 1989
*
* Written for the MPW IIGS Assembler -- Version 1.1b1, 4/9/90
*
* Copyright 1989-1990 by Apple Computer, Inc.
*
***************************************************************************

      case yes
      machine M65816
      string asis
      msb on
      print on

      export CheckPatch              ; Check for a valid patch header.
      export InstallE10000           ; Adds a patch into the toolbox vectors.
      export RemoveE10000            ; Removes a toolbox dispatch vector patch.
      export PatchHeader             ; The simplest toolbox dispatch vector patch.
```

```
****************************************************************************
* Equates - Various equates required by these routines.
*
versionNumber         equ $0100          ; The version number of this library.

dispatch1             equ $e10000        ; The first toolbox dispatch vector.
dispatch2             equ $e10004        ; The second toolbox dispatch vector.
ToolPointerTable      equ $e103c0        ; Pointer to the active System TPT.
UserToolPointerTable  equ $e103c8        ; Pointer to the active User TPT.

; Error return values from the routines InstallE10000 and RemoveE10000

noError               equ $0000          ; Value returned if no error occurs.
badHeaderError        equ $8001          ; Patch header wasn't valid.
headerNotFoundError   equ $8002          ; Header to remove wasn't in the linked list.
```

`PatchHeader` is the standard shell for the actual patch code. Your code goes in here, at `NewDispatch2`. When you get control at `NewDispatch2`, the function number is in X and there are two RTL addresses on the stack (pushed after the function's parameters).

Your patch code does not care whether the tool call is being made through the $E10000 or $E10004 vector—in either case you get control with two `RTL` addresses on the stack.

```
****************************************************************************
* PatchHeader - Header required of all routines that will be patched
*                into the toolbox dispatch vectors.
*
* Note:  The code between next1Vector and NewDispatch2 must be included
*        for all calls.  The code below NewDispatch2 only needs to be
*        included for patches that want to post patch the calls.
*
PatchHeader proc
      entry next1Vector,next2Vector
      entry dispatch1Vector,dispatch2Vector
      entry NewDispatch1,NewDispatch2

next1Vector                              ; Where dispatch1 should go when finished.
      jml next1Vector                   ; (Filled in by InstallE10000).
next2Vector                              ; Where dispatch2 should go when finished.
      jml next2Vector                   ; (Filled in by InstallE10000).
dispatch1Vector                          ; Holds the JML instruction from $e10000.
      jml dispatch1Vector               ; (Filled in by InstallE10000).
dispatch2Vector                          ; Holds the JML instruction from $e10004.
      jml dispatch2Vector               ; (Filled in by InstallE10000).

anRtl  rtl                               ; An RTL instruction.  Its address will be
                                        ; pushed on the stack for dispatch1 calls.

NewDispatch1                             ; Entry point for dispatch1 toolbox vector.

      phk                               ; Push program bank.
      pea anRtl-1                       ; Push the address of a RTL.

NewDispatch2                             ; Entry point for dispatch2 toolbox vector.

; The following code should be included in the PatchHeader if the patch wants
; to perform post patching.  This code will determine if the call that was made
; actually exists and if it does, post patching can occur.  If the call doesn't
; exist, any pre-call routines can be executed, but the post patching shouldn't
; be attempted because the dispatcher will remove the second return address from
; the stack, thus not returning to your post patching routines.
; Stack equates for this routine.
```

```
aLong  equ $0001                   ; A temporary long value.
oldDP  equ aLong+4                 ; Where the direct page is saved to.
oldTM  equ oldDP+2                 ; Where the tool call number is saved.

      phx                         ; Save the call that's being made.
      phd                         ; Save the current direct page.
      lda >ToolPointerTable+2     ; Get the TPT to determine the number
      pha                         ; of tool sets loaded.
      lda >ToolPointerTable
      pha
      tsc                         ; Set the direct page to the stack.
      tcd
      txa                         ; See if this tool set exits.
      and #$00ff
      cmp [aLong]                 ; Is it larger than the number of tool sets?
      bcs @noCall                 ; JIF this tool set doesn't exist.
      asl a
      asl a
      tay                         ; Now get the pointer to the FPT.
      lda [aLong],y
      tax
      iny
      iny
      lda [aLong],y
      sta aLong+2
      stx aLong
      lda oldTM                   ; Get the function number.
      and #$ff00
      xba
      cmp [aLong]                 ; Compare it to the number of entries in table.
@noCall
      pla                         ; Remove aLong from the stack.
      pla
      pld                         ; Restore the original direct page.
      plx                         ; Recover the tool number.

; At this point the carry flag is set if the tool call doesn't exist and clear
; if the tool call exits.  No post patching must occur if the carry flag is set.

      jmp next2Vector             ; Go to the original $e10004 jump instruction.

      endp

****************************************************************************
* CheckPatch - Checks the passed toolbox dispatch vector to see if it
*              points to a valid patch.
*
* Input: Passed via the stack following C conventions.
*     newPatchAddr (long) - Address of the patch routine.
*
* Output:
*     If newPatchAddr is a valid patch -
*          Carry clear
*     If newPatchAddr is not a valid patch -
*          Carry set
*
CheckPatch proc

zprtl         equ $01              ; The address for the rtl on our direct page.
newPatchAddr  equ zprtl+3          ; Address of patch (parameter to this routine).

      tsc                         ; Make the stack the direct page after saving
      phd                         ; the current direct page.
      tcd
      lda newPatchAddr+2          ; Simple check to check for a valid pointer.
      and #$ff00
      bne BadPatch                ; Wasn't zero, can't be a valid pointer.

      lda [newPatchAddr]          ; Check for the first JML instruction.
      and #$00ff
      cmp #$005c
      bne BadPatch

      ldy #$04                    ; Check for the second JML instruction.
      lda [newPatchAddr],y
      and #$00ff
      cmp #$005c
      bne BadPatch

      ldy #$08                    ; Check for the third JML instruction.
      lda [newPatchAddr],y
      and #$00ff
      cmp #$005c
      bne BadPatch

      ldy #$0c                    ; Check for the fourth JML instruction.
      lda [newPatchAddr],y
      and #$00ff
      cmp #$005c
      bne BadPatch

      ldy #$10                    ; Check for the rtl and phk instructions.
      lda [newPatchAddr],y
      cmp #$4b6b
      bne BadPatch

      iny                         ; Check for the phk and pea instructions.
      lda [newPatchAddr],y
      cmp #$f44b
      bne BadPatch

      clc                         ; Calculate the address of the rtl instruction.
      lda newPatchAddr
      adc #$000f
      ldy #$13                    ; Check for address of the rtl instruction.
      cmp [newPatchAddr],y
      bne BadPatch

GoodPatch
      pld                         ; Restore the direct page and report
      clc                         ; that it was a good patch.
      rtl

BadPatch
      pld                         ; Restore the direct page and report
      sec                         ; that something was wrong.
      rtl

      endp

****************************************************************************
* InstallE10000   - Sets the jump vector at $e10000 and $e10004 to point to
*                 the passed new toolbox dispatch vector patch.  This routine
*                 also updates the linked lists so that more than one routine
*                 can be patched into the dispatch vectors.
*
* Input: Passed via the stack following C conventions.
*     newPatchAddr (long) - Address of the patch routine.
*
```

```
* Output:
*     If an error occurred -
*          Carry set, Accumulator contains one of the following error codes:
*               badHeaderError
*     If no error occurred and patch was installed successfully -
*          Carry clear, Accumulator contains zero.
*
InstallE10000 proc

oldPatchAddr    equ $01                   ; Address of existing patch.
zprtl           equ oldPatchAddr+4        ; The address for the rtl.
zpsize          equ zprtl-oldPatchAddr    ; Size of direct page we'll have on the stack.
newPatchAddr    equ zprtl+3               ; Address of patch (parameter to this routine).

      tsc                          ; Move the stack pointer to point beyond
      sec                          ; the direct page variables that we'll
      sbc #zpsize                  ; place on the stack.
      tcs
      phd                          ; Save the direct page register.
      tcd                          ; Set the direct page.
      php                          ; Disable interrupts
      sei

      pei newPatchAddr+2           ; Check if patch header is valid.
      pei newPatchAddr
      jsl CheckPatch
      plx                          ; Remove the parameters from the stack.
      plx
      bcc @1                       ; Report the badHeaderError if detected.
      ldy #badHeaderError
      jmp Exit

@1     lda >dispatch1               ; Set up the next1Vector in the new patch.
      sta [newPatchAddr]           ; The JML instruction and low byte.
      lda >dispatch1+2
      ldy #$02
      sta [newPatchAddr],y         ; The middle and upper bytes.

      lda >dispatch2               ; Set up the next2Vector in the new patch.
      ldy #$04
      sta [newPatchAddr],y         ; The JML instruction and low byte.
      lda >dispatch2+2
      ldy #$06
      sta [newPatchAddr],y         ; The middle and upper bytes.

      lda >dispatch1+3             ; See if there's already a patch in dispatch1.
      and #$00ff
      sta oldPatchAddr+2
      pha                          ; High byte of possible header address.
      lda >dispatch1+1
      sec
      sbc #$0011
      sta oldPatchAddr
      pha                          ; Low byte of possible header address.
      jsl CheckPatch
      plx
      plx
      bcs First                    ; JIF this will be the first patch installed.

      ldy #$08                     ; Set up the dispatch1Vector in the new patch.
      lda [oldPatchAddr],y
      sta [newPatchAddr],y         ; The JML instruction and low byte.
      ldy #$0a
      lda [oldPatchAddr],y
      sta [newPatchAddr],y         ; The middle and upper bytes.
      ldy #$0c                   ; Set up the dispatch2Vector in the new patch.
      lda [oldPatchAddr],y
      sta [newPatchAddr],y       ; The JML instruction and low byte.
      ldy #$0e
      lda [oldPatchAddr],y
      sta [newPatchAddr],y       ; The middle and upper bytes.

      bra PatchIt                ; Now patch dispatch1 and dispatch2.

First  ldy #$08                   ; Set up the dispatch1Vector in the new patch.
      lda >dispatch1
      sta [newPatchAddr],y       ; The JML instruction and low byte.
      ldy #$0a
      lda >dispatch1+2
      sta [newPatchAddr],y       ; The middle and upper bytes.

      ldy #$0c                   ; Set up the dispatch2Vector in the new patch.
      lda >dispatch2
      sta [newPatchAddr],y       ; The JML instruction and low byte.
      ldy #$0e
      lda >dispatch2+2
      sta [newPatchAddr],y       ; The middle and upper bytes.

PatchIt
      clc                        ; Calculate the address of the new dispatch2.
      lda newPatchAddr
      adc #$0015
      sta newPatchAddr
      xba
      and #$ff00                 ; Mask in the JML instruction.
      ora #$005c
      sta >dispatch2             ; The JML instruction and low byte.
      lda newPatchAddr+1
      sta >dispatch2+2           ; The middle and upper bytes.

      sec                        ; Calculate the address of the new dispatch1.
      lda newPatchAddr
      sbc #$0004
      sta newPatchAddr
      xba
      and #$ff00                 ; Mask in the JML instruction.
      ora #$005c
      sta >dispatch1             ; The JML instruction and low byte.
      lda newPatchAddr+1
      sta >dispatch1+2           ; The middle and upper bytes.

      ldy #noError               ; Report that all went well.

Exit   plp                        ; Restore the interrupt state.
      pld                        ; Restore the previous direct page register.
      tsc                        ; Restore the stack pointer.
      clc
      adc #zpsize
      tcs
      tya                        ; Value to return.
      beq @noerr
      sec                        ; Report that there was an error.
      rtl
@noerr clc                        ; Report that there was no error.
      rtl

      endp
```

```
****************************************************************************
* RemoveE10000 - Removes the specified patch from the dispatch1 and dispatch2
*                vectors and updates the linked lists for the remaining
*                toolbox patches.
*
* Input: Passed via the stack following C conventions.
*     patchToRemove (long) - Address of the patch to remove.
*
* Output:
*     If an error occurred -
*          Carry set, Accumulator contains one of the following error codes:
*               badHeaderError
*               headerNotFoundError
*     If no error occurred and patch was removed successfully -
*          Carry clear, Accumulator contains zero.
*
RemoveE10000 proc

patchDispAddr    equ $01                    ; Address of existing patch (and 1 extra byte).
prevHeader       equ patchDispAddr+5        ; Used to search through the linked list.
zprtl            equ prevHeader+4           ; The address for the rtl.
zpsize           equ zprtl-patchDispAddr    ; Size of direct page we'll have on the stack.
patchToRemove    equ zprtl+3                ; Address of patch (parameter to this routine).

      tsc                           ; Move the stack pointer to point beyond
      sec                           ; the direct page variables that we'll
      sbc #zpsize                   ; place on the stack.
      tcs
      phd                           ; Save the direct page register.
      tcd                           ; Set the direct page.
      php                           ; Disable interrupts
      sei

      pei patchToRemove+2           ; Check if patch header we were asked to
      pei patchToRemove             ; remove is a valid header.
      jsl CheckPatch
      plx                           ; Remove the parameters from the stack.
      plx
      bcc @1                        ; Report the badHeaderError if detected.
      ldy #badHeaderError
      jmp Exit

@1     clc                           ; Create the JML instruction that would exist
      lda patchToRemove             ; if the patchToRemove was installed.
      adc #$0011
      sta patchDispAddr+1
      lda patchToRemove+2
      sta patchDispAddr+3
      lda patchDispAddr             ; Mask in the JML instruction.
      and #$ff00
      ora #$005c
      sta patchDispAddr

      cmp >dispatch1                ; Check if the patch to remove is the first
      bne NotFirstOne               ; patch installed.
      lda >dispatch1+2
      cmp patchDispAddr+2
      bne NotFirstOne

      lda [patchToRemove]           ; Restore the Dispatch1 vector.
      sta >dispatch1
      ldy #$02
      lda [patchToRemove],y
      sta >dispatch1+2
      ldy #$04                    ; Restore the Dispatch2 vector.
      lda [patchToRemove],y
      sta >dispatch2
      ldy #$06
      lda [patchToRemove],y
      sta >dispatch2+2

      bra NoErr                   ; Everything went well.

NotFirstOne
      sec                         ; Assume that whatever is in dispatch1 is a
      lda >dispatch1+1            ; patch and get the address of its header.
      sbc #$0011
      sta prevHeader              ; Low and middle bytes.
      lda >dispatch1+3
      and #$00ff
      sta prevHeader+2            ; Upper byte of header address.

@loop  pei prevHeader+2            ; Check if it really is a valid header.
      pei prevHeader
      jsl CheckPatch
      plx                         ; Remove the parameters from the stack.
      plx
      bcc @2                      ; Report that the patch that we asked to
      ldy #headerNotFoundError    ; remove wasn't found.
      bra Exit

@2     lda [prevHeader]            ; See if this patch points to patch we want
      cmp patchDispAddr           ; to remove.
      bne @nope
      ldy #$02
      lda [prevHeader],y
      cmp patchDispAddr+2
      bne @nope

      lda [patchToRemove]         ; Restore the next1Vector.
      sta [prevHeader]
      ldy #$02
      lda [patchToRemove],y
      sta [prevHeader],y

      ldy #$04                    ; Restore the next2Vector.
      lda [patchToRemove],y
      sta [prevHeader],y
      ldy #$06
      lda [patchToRemove],y
      sta [prevHeader],y

      bra NoErr                   ; Everything went well.

@nope  ldy #$02                    ; Get the address of the next patch header.
      lda [prevHeader],y
      tax
      lda [prevHeader]
      sta prevHeader
      stx prevHeader+2

      sec
      lda prevHeader+1
      sbc #$11
      sta prevHeader
      lda prevHeader+3
      and #$00ff
      sta prevHeader+2
      bra @loop                    ; Now check this header.

NoErr  ldy #noError                 ; Report that all went well.

Exit   plp                          ; Restore the interrupt state.
      pld                          ; Restore the previous direct page register.
      tsc                          ; Restore the stack pointer.
      clc
      adc #zpsize
      tcs
      tya                          ; Value to return.
      beq @noerr
      sec                          ; Report that there was an error.
      rtl
@noerr clc                          ; Report that there was no error.
      rtl

      endp

      end
```

Further Reference

- *Apple IIGS Toolbox Reference*
- Apple IIGS Technical Note #73, Using User Tool Sets