New Technical Notes
Macintosh

Developer Support

Key Mapping
Toolbox

Revised by: Jim Luther, Peter Edberg, & Imran Sayeed February 1991
Written by: Cameron Birse September 1987

This Technical Note describes the Macintosh family key code mapping scheme when running System file 4.1 and later. This Note also provides a "safe" method for remapping keyboards.
Changes since October 1990: Added a section on how `'KMAP'` resources are matched to specific ADB keyboard types and a section on the original Macintosh and Macintosh Plus keyboards.

Introduction

System file 4.1 introduced a change in the keystroke mapping mechanism for the Macintosh family. Originally, a keystroke caused an interrupt, and the interrupt handler dispatched the keycode to a translation routine pointed to by a low-memory pointer (`Key1Trans` or `Key2Trans`). The System used to install this routine at boot time, and developers would generally replace it in part or entirely to remap the keyboard. When the keycode was mapped, it was returned to the interrupt handler, which then posted the event. The System file contained both the translation routine and the key map in `'INIT'` resources `ID = 0` and `ID = 1`.

In all System files since 4.1, the low-memory pointers are still there, and the Macintosh Plus still calls them; however, Macintosh systems equipped with ADB do not call these low-memory pointers. The System preserves them so applications that call them can still use them to translate keycodes, but since System file 4.1, they point to a routine that implements a different mechanism.

ADB Keyboards

With multiple Apple Desktop Bus (ADB) keyboards, a mechanism is needed to map the different raw keycodes to a standard virtual keycode that can be mapped to ASCII and special character sets. This mapping is done in an effort to reduce keyboard hardware dependence, and the raw mapping routine uses a table which is resident in the System `'KMAP'` resource. Basically, the raw keycode is used to index into the `'KMAP'` table; the value at the indexed location in the `'KMAP'` is what gets returned as the virtual keycode.

The `'KMAP'` resource ID that matches the keyboard type is used if that `'KMAP'` resource is present. If a `'KMAP'` resource ID that matches the keyboard type cannot be found, then the system attempts to use `'KMAP'` resource `ID = 0`. If `'KMAP'` resource `ID = 0` cannot be found, then raw keycode to virtual keycode mapping does not occur. On all Macintosh systems later than the Macintosh Plus, `'KMAP'` resource `ID = 0` is in ROM. The global variable `KbdType` (a byte) contains the type of the last keyboard used.

KbdType	Keyboard Type
$01	Apple Keyboard (standard) and Apple Desktop Bus Keyboard (IIGS)
$02	Apple Extended Keyboard and Apple Extended Keyboard II
$04	Apple Keyboard, ISO
$05	Apple Extended Keyboard II, ISO
$06	Portable
$07	Portable, ISO
$08	Apple Keyboard II
$09	Apple Keyboard II, ISO

With this mechanism, the keystroke causes an interrupt, and the interrupt handler maps the raw keycode to a "virtual" keycode , which is then sent to the `_KeyTrans` trap. This trap maps the virtual keycode to an ASCII value (using tables which reside in the `'KCHR'` resource of the System file) and returns that value to the handler which posts the event.

```
FUNCTION KeyTrans(transData:Ptr;keycode:INTEGER;VAR state: LONGINT):LONGINT
```

The `transData` parameter is a pointer to the `'KCHR'` image in memory. The `keycode` parameter is an integer composed of the modifier flags in bits 8-15, an up or down stroke flag in bit 7 (1=up), and the virtual key code in bits 6-0. The `state` parameter is a value internal to `_KeyTrans` which should be preserved across calls if dead keys are desired. It is dependent on the `'KCHR'` information, so if the `'KCHR'` is changed, state should be reset to zero.

The `LONGINT` returned is actually two 16-bit characters to be posted as events (usually the high byte of each is zero), high word first. A returned value of zero in either word should not be posted. Do not depend upon the word in which the character is returned; if both words are valid, then the high word should be posted first.

To remap the keyboard, one must supply a `'KCHR'` resource and have the System use it. Each `'KCHR'` resource has an associated `'SICN'` resource. The `'SICN'` resource provides a graphic representation of the current keyboard mapping. For example, the French keyboard layout has a `'SICN'` of a French flag to designate that particular map is currently active. The `'SICN'` resource should be some representation of the particular remap, and its ID number must be the same as that of the `'KCHR'`. The `'KCHR'` resource must be named appropriately, as it can be displayed in a scrolling list in the Keyboard Control Panel.

Macintosh and Macintosh Plus Keyboards

With the Macintosh (the original keyboard on the 128K and 512K Macintoshes) and Macintosh Plus keyboards, the event record contains the raw keycode, since there is no `'KMAP'` mapping. For the domestic Macintosh keyboard and the Macintosh Plus keyboard, this is not a problem, since the raw keycodes generated by those keyboards are identical to the virtual keycodes. This is not the case for the Macintosh international keyboard, which is still used with the Macintosh Plus on many international systems. For this keyboard, the event record contains a raw keycode which can not be treated as a virtual keycode.

If you need to obtain virtual keycodes for the Macintosh international keyboard, you need to map the raw keycode in the event record to a virtual keycode. The following table provides the necessary mapping (the raw keycodes generated by this keyboard are in the range $00-$3F; keycodes above this are generated by the optional keypad that may be used with this keyboard). If the raw keycode is used as an offset into this table, the byte at that offset is the virtual keycode. This mapping is also performed by the `_Key1Trans` hook before it calls `_KeyTrans`, if the keyboard is a Macintosh international type.

```
oldIntlKeybdRawToVirtual                                  ; raw keycode:
        dc.b  $00, $01, $02, $03, $04, $05, $32, $06   ; $00 .. $07
        dc.b  $07, $08, $2c, $09, $0c, $0d, $0e, $0f   ; $08 .. $0f
        dc.b  $10, $11, $12, $13, $14, $15, $16, $17   ; $10 .. $17
        dc.b  $18, $19, $1a, $1b, $1c, $1d, $1e, $1f   ; $18 .. $1f
        dc.b  $20, $21, $22, $23, $2a, $25, $26, $27   ; $20 .. $27
        dc.b  $28, $29, $24, $2e, $2f, $0b, $2d, $2b   ; $28 .. $2f
        dc.b  $30, $34, $0a, $33, $31, $35, $36, $37   ; $30 .. $37
        dc.b  $38, $39, $3a, $3b, $3c, $3d, $3e, $3f   ; $38 .. $3f
```

The global variable `KbdType` (a byte) contains the type of the last keyboard used. The following table shows the values of the global variable `KbdType` for the Macintosh and Macintosh Plus keyboards:

KbdType	**Keyboard Type**	
	$03	Macintosh (domestic or international)
	$0B	Macintosh Plus

For both the Macintosh domestic and international keyboards, the global variable `KbdType` is 3. The Macintosh has no way to distinguish between these two keyboards, and must rely on the user to indicate which keyboard is being used by clicking on the appropriate picture in a panel in the Keyboard Cdev (this panel only appears on non-ADB Macintoshes). A byte flag in the `'itlc'` resource `ID = 0` indicates which keyboard the user has specified. If the `KbdType` global contains 3, you can test this flag to determine if the international version of the keyboard is being used (following is the assembly-language version):

```
with      ItlcRecord
subq      #4,sp               ; space for returned handle
move.l    #'itlc',-(sp)       ; push itlc type
clr.w     -(sp)               ; want ID=0
_GetResource                  ; get the itlc resource
move.l    (sp)+,d0            ; did we get it?
beq       myErrorHandling     ; if not, bail
move.l    d0,a0               ; copy handle
move.l    (a0),a0             ; get pointer
tst.b     itlcOldKybd(a0)     ; check flag
endwith  ;ItlcRecord
; if non-zero, international keyboard is being used
```

Hardware Dependencies

Although the principle underlying virtual keycodes is to have a standard keycode for a character regardless of the actual keyboard used, some hardware dependent differences are still present, and covered in this section.

- The virtual keycodes for the cursor keys and for some keypad operator keys are different on the ADB keyboards and the non-ADB keyboards:

Key Description	ADB Keycode	Non-ADB Keycode
left arrow	$7B	$46
right arrow	$7C	$42
down arrow	$7D	$48
up arrow	$7E	$4D
keypad plus sign (+)	$45	$46 (with Shift bit set in modifiers)
keypad asterisk (*)	$43	$42 (with Shift bit set in modifiers)
keypad equal sign (=)	$51	$48 (with Shift bit set in modifiers)
keypad slash (/)	$4B	$4D (with Shift bit set in modifiers)

Notice that on non-ADB keyboards, the keycodes for the keypad operators listed duplicate the keycodes for the cursor keys. On these keyboards, holding Shift and pressing Left Arrow produce the plus sign character (+), for example.

- The Macintosh International keyboard and the ISO ADB keyboards have an extra key that is not present on the domestic keyboards. This key produces virtual keycode $0A.

- It is possible to reassign the virtual keycodes for the Shift, Option, and Control keys on the right side of the ADB Extended keyboards. Please refer to *Inside Macintosh*, V-193, The Toolbox Event Manager, for an explanation.

- There is a different virtual keycode for the Enter key depending on whether it is on the keypad (as on the Macintosh Plus keyboard and most ADB keyboards) or on the main section of the keyboard (as on the Macintosh keyboard and the Portable keyboard). The keypad version has keycode $4C, while the main keyboard version has keycode $34.

Remapping the Keyboard

Remapping the keyboard can be done two ways: either at boot time or from within an application. Remapping from within an application can be made permanent (until the next boot) or only for the life of the application. The remapping is accomplished by modifying a `'KCHR'` resource and telling the System to use the new `'KCHR'`. The `'KCHR'` must have an ID number in the range of the appropriate script and must have an associated `'SICN'` resource with the same ID number. The Roman script, for example, uses the range 0 to 16383, and the standard `'KCHR'` IDs for each country are the same as the country code (e.g., US = 0, French = 1, German = 2, etc.). Other Roman keyboards should have numbers somewhere in the script range (e.g., “Dvorak” at ID 500).

Remap At Boot Time

To remap the keyboard at boot time, there must be a modified `'KCHR'` resource in the System file with an ID number in the range of the appropriate script and an associated `'SICN'` resource with the same ID number. To make the System use the modified `'KCHR'` at boot time, one must change the entries in the `'itlb'` resource in the System file to reflect the ID of the modified 'KCHR' and 'SICN' resources. Refer to the latest version of MPW's SysTypes.r file for the exact format of the 'itlb' resource.

Since Apple's System Software Licensing policy forbids shipping a modified System file, Apple suggests using the Installer to install the new resources. This installation should assign the new resources their IDs based upon what is currently in the System file to avoid all conflicts. Refer to M.PT.Installer for more details on Apple's Installer program.

Remap After Boot Time

To remap the keyboard after boot time, there must be a modified 'KCHR' resource in the System file or in the application with an ID number in the range of the appropriate script and an associated 'SICN' resource with the same ID number. One must also call the Script Manager to set up the proper resources and tell the System to use them. First call _SetScript to set the Script Manager's global variable for the 'KCHR' resource ID, then call _SetScript again to set the global variable for the 'SICN' resource ID. Now call _KeyScript to load the resources and set up the System to use them. The following sample code demonstrates these calls:

MPW Pascal

```
CONST
  DvorakID = 500;

VAR
  err: OSErr;

BEGIN
  err := SetScript(smRoman, smScriptKeys, DvorakID);
  err := SetScript(smRoman, smScriptIcon, DvorakID);
   KeyScript(smRoman);
END;
```

'KCHR' Resource Format

This section provides a general description of the 'KCHR' resource format; refer to the latest version of MPW's SysTypes.r file for the exact format of the 'KCHR' resource.

The 'KCHR' resource consists of a two-byte version number followed by a 256-byte modifier table, mapping all 256 possible modifier states to a table number. This table is followed by a two-byte count of tables, which is, in turn, followed by that many 128-byte ASCII tables. The ASCII tables map the virtual keycode to an ASCII value; zero signifies a dead key, and in this case the dead key table must be searched. The dead key table is composed of a count of dead key records (two bytes) and that many dead key records. A dead key record consists of a one-byte table number, a one-byte virtual keycode (without an up or down bit), a completor table, and a no-match character.

When _KeyTrans searches the dead key records, it checks for a match with the table number and the keycode. If there is no match, it is not a dead key, and a zero is returned. If there is a match, it is recorded in the state variable. If the previous key was a dead key, the completor table is searched. The completor table is comprised of a count of completor records, followed by that number of completor records.

A completor record is simply a substitution list for ASCII characters. If the ASCII character matches the first byte in the completor record, the second byte is substituted for it. When there is no substitution to be made, the original ASCII character is preceded by the no match character found at the at the end of the dead key record.

'KMAP' Resource Format

This section provides a general description of the `'KMAP'` resource format; refer to the latest version of MPW's SysTypes.r file for the exact format of the `'KMAP'` resource.

The 'KMAP' resource starts with a two-byte ID, followed by a two-byte version number. These four bytes are followed by the 128-byte keycode mapping table, described previously. The table is followed by a list of exceptions. The 128-byte table is simply a one-to-one mapping of real keycodes to virtual keycodes; the first byte is the virtual keycode for $00, the second for $01, etc. The high bit of the virtual keycode signals an exception entry in the exception list.

The exception list is used to enable the device driver to initiate communication with the device, usually to perform a state change. The exception list begins with a two-byte record count followed by that many records. The format of the exception record is available in the MPW SysTypes.r file. The raw keycode is the keycode as generated by the device. The `XOR` bit informs the driver to invert the state of the key instead of using the state provided by the hardware. This can be used to provide keys that lock in software. *Inside Macintosh*, Volume V, The Apple Desktop Bus, describes the ADB opcode. Finally, the data string is a Pascal string that is passed to the `_ADBOp` trap.

Further Reference:

- *Inside Macintosh*, Volume V, The Toolbox Event Manager
- *Inside Macintosh*, Volume V, The Apple Desktop Bus
- M.PT.Installer
- M.TE.InternationalCancel