Macintosh Technical Notes

#160: Key Mapping

See also: The Script Manager

Written by: Cameron Birse September 1, 1987
Updated: March 1, 1988

This technical note describes the key code mapping scheme used in the Macintosh SE, Macintosh II, and System file 4.1 and later. It also provides a “safe” method for remapping keyboards.

The keystroke mapping mechanism has been changed for the Macintosh SE, the Macintosh II, and version 4.1 of the System file. 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`). This routine was installed at boot time by the System, and was generally replaced entirely or in part for remapping the keyboard. When the keycode was mapped, it was returned to the interrupt handler, and the handler then posted the event. The translation routine and the key map were both contained in the System file resources INIT 0 and 1.

In the new System file, the low-memory pointers are still there and are still called by the Macintosh Plus, but are not called by ADB systems. They are preserved so that applications that call them will still be able to use them to translate keycodes. They now point to a routine that implements the new System mechanism. In this new mechanism, the keystroke causes an interrupt, and the interrupt handler maps the raw keycode to a “virtual” keycode, which is then sent to a trap called `KeyTrans`. This routine maps the virtual keycode to an ASCII value, and returns that to the handler, which posts the event.

With the advent of new keyboards, a mechanism was needed to map the different raw keycodes to a standard virtual keycode that could be mapped to the ASCII and special character sets. The mapping from raw to virtual keycodes is done for keyboard hardware independence. The raw mapping routine uses a table resident in the System resource KMAP. Basically, the raw keycode is used to index into the table in KMAP. The value at the indexed location in KMAP is what is returned as the virtual keycode. The format of the KMAP resource is described below.

The mapping of the virtual keycode to the ASCII character is done by a new trap in the Macintosh SE and Macintosh II ROMs, and in System 4.1 (and later) for the Macintosh Plus. The new trap is called `KeyTrans` (not to be confused with the `Key1Trans` or `Key2Trans` pointers), and it maps the virtual keycode to an ASCII keycode (based on modifiers, if any) using tables that reside in the System resource KCHR. The format of the KCHR resource is also described below.

```
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 comprised of the modifier flags in bits 8-15, an up/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 0.

The `LONGINT` returned is actually two 16-bit characters to be posted as events (usually the high byte of each is 0), high word first. A returned value of 0 in either word should not be posted. Do not depend on which word the character will end up in. If both words are valid, then the hi word should be posted first.

In the Macintosh SE and Macintosh II, the KMAP and KCHR resources are in ROM. In the Macintosh Plus, the System uses a KCHR resource in the System file. In order to remap the keyboard, you must supply a KCHR resource and have the System use it. In addition to the KCHR resource, there is 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 your particular remap, and its ID# must be the same as the KCHR's. The KCHR resource must be named appropriately, as there will be a Control Panel option that will allow the user to choose the appropriate map.

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 (US = 0, French = 1, German = 2, Italian = 3, etc.). Alternative Roman keyboards should have numbers somewhere in the script range, e.g. "Dvorak" at ID 500.

To remap the keyboard at boot time, there must be a modified KCHR resource in the System file with an ID in the range of the appropriate script. In addition to the KCHR resource, there must be an associated SICN resource. To make the System use the modified KCHR at boot time, you must change the entries in the itlb resource in the System file to reflect the ID of the modified KCHR, and SICN resources. The `itlb` resource format is described below.

Apple's System Software Licensing policy forbids shipping a modified System file; we suggest an installer program that the user can execute to perform the installation of the new resources. The installer program should assign the new resources their IDs based on what is currently in the System file so there won't be a conflict.

itlb resource format

```
/*-----------------itlb • International Script Bundle----------------*/
type 'itlb' {
unsigned integer;   /* itl0 id number                                  */
unsigned integer;   /* itl1 id number                                  */
unsigned integer;   /* itl2 id number                                  */
unsigned integer;   /* reserved                                        */
unsigned integer;   /* reserved                                        */
unsigned integer;   /* reserved                                        */
unsigned integer;   /* language code {e.g. langFrench}                 */
unsigned integer;   /* number/date codes                               */
unsigned integer;   /* KCHR id number                                  */
unsigned integer;   /* SICN id number                                  */
};
```

To remap the keyboard after boot, you need a KCHR (and SICN) in your application or in the System file (with an ID in the range of the appropriate script), and you need to use the following calls to the Script Manager (the Script Manager is included in System 4.1 and later). The first call to `SetScript` sets the Script Manager's global variable for the KCHR resource ID, and the second call to `SetScript` sets the Script Manager's global variable for the SICN resource ID. When these calls are completed, the call to `keyScript` will load the resources, and set up the System to use them.

```
CONST
   DvorakID = 500;

VAR
   err: OSErr;

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

KCHR resource format

```
Version                                    2 bytes (integer)
Indexes (to character table)               256 bytes (array of bytes)
Count of character table arrays            2 bytes (integer)
Character table arrays                     128*n bytes (array of char)
Count of DeadKey Records                   2 bytes (integer)
   table number                            1 byte
   virtual keycode                         1 byte
   count of Completor Records              2 bytes (integer)
      completor character                  1 byte (char)
      substituting character               1 byte (char)
   no match character                      1 byte (char)
Extra room for 16 bit Character Codes      1 byte (char)
```

The KCHR resource consists of a 2 byte version number followed by a 256 byte modifier table, mapping all 256 possible modifier states to a table number. This is followed by a 2 byte count of tables, which is, in turn, followed by that many 128 byte ASCII tables. The ASCII table maps the virtual keycode to an ASCII value; 0 signifies a dead key, in this case the dead key table must be searched.

The dead key table is comprised of a count of dead key records (2 bytes), and that many dead key records. A dead key record consists of a one byte table number, a one byte virtual keycode (without up/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 no 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 will be 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 will be 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

```
ID                                  2 bytes (integer)
Version                             2 bytes (integer)
Raw to virtual keycode map          128 bytes (array)
Count of Exception arrays           2 bytes (integer)
Exception arrays                    (array)
   raw keycode                      1 byte
   noXor, Xor                       1 bit (boolean)
   fill bits                        3 bits
   ADB opcode                       4 bits (bitstring)
   data string                      variable (Pascal string)
```

The KMAP resource starts with a two byte ID, followed by a two byte version number. This is followed by the 128 byte keycode mapping table, described above in this technical note. 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 shown above. 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. The ADB opcode is described in the *Inside Macintosh* chapter on the Apple DeskTop Bus in Volume V. Finally the data string is a Pascal string that is passed to the `ADBOp` trap.