Keyboard Resources

This appendix describes the Macintosh keyboard resources. The keyboard resources make text input possible; they provide a hardware interface to different types of keyboards and a software interface to different script systems. Some of the keyboard resources belong to an individual script system and are independent of any particular keyboard; others belong to a type of keyboard and are independent of any script system.

By installing the appropriate keyboard resources, you can perform text input in any script system, from any Macintosh-supported keyboard. By modifying the keyboard resources, you can localize or customize text input by changing keyboard layouts, remapping key combinations, creating keyboard icons, modifying the keyboard-layout display in the Key Caps desk accessory, and changing the way the user switches among keyboard layouts and keyboard scripts.

Most text-processing applications have no need for direct access to keyboard resources. You may need to read this appendix, however, if you are

- using the Event Manager `KeyTranslate` function to get specific information from a custom keyboard-layout resource or to make Command-key handling more script-independent
- creating your own localized version of a script system
- designing a new type of keyboard

Before reading this appendix, read the chapter "Introduction to Text on the Macintosh" in this book. The keyboard resources are used by the Script Manager, described in this book, and by the Event Manager and Menu Manager, described in *Inside Macintosh: Macintosh Toolbox Essentials.* Resources in general are described in the Resource Manager chapter of *Inside Macintosh: More Macintosh Toolbox.* Additional information on keyboards themselves can be found in *Inside Macintosh: Devices* and in *Guide to the Macintosh Family Hardware.*

This appendix starts with a brief discussion of keyboards. It then lists the keyboard resources, shows how they may differ in different versions of localized software, and presents the concept of key translation. It then discusses each keyboard resource in detail.

Note
All keyboard information that relates to virtual key codes and their relation to raw key codes is discussed under "Key-Map Resource (Type 'KMAP')" beginning on page C-11, even if it is not specifically related to the key-map resource. ◆

About Keyboards

The Macintosh computer supports over 12 separate physical types of keyboards. Your application needs to be able to handle text input from the domestic and ISO layouts of all Apple keyboards. It also needs to be able to distinguish multiple keyboards and to use the modifier flag that detects the state of the **modifier keys** (Shift, Caps Lock, Command, Option, and Control) on keyboards.

Table C-1 lists the keyboard types. These type values are used in some of the keyboard resources discussed later in this appendix.

Table C-1 The keyboard types

Keyboard type*	Keyboard
1	Apple Keyboard and Apple Keyboard II (domestic layout)
2	Apple Extended Keyboard and Apple Extended Keyboard II (domestic layout)
3	Small Macintosh 512K Keyboard (no keypad; domestic layout)
4	Apple Keyboard (ISO layout)
5	Apple Extended Keyboard II (ISO layout)
6	Apple Macintosh Portable Keyboard (domestic layout)
7	Apple Macintosh Portable Keyboard (ISO layout)
8	Apple Macintosh Keyboard II (domestic layout)
9	Apple Macintosh Keyboard II (ISO layout)
11	Macintosh Plus Keyboard with the built-in keypad
12	Macintosh PowerBook Keyboard (domestic layout)
13	Macintosh PowerBook Keyboard (ISO layout)
259	Small Macintosh 512K Keyboard (no keypad; ISO layout)

* Keyboard type is also the resource ID of the corresponding `'KMAP'` or `'KCAP'` resource. The `KbdType` low-memory global variable contains the low byte of this value for the last keyboard used.

Figure C-1 and Figure C-2 show the U.S. layout of the Apple Keyboard II and Apple Extended Keyboard II and the virtual key codes produced by each key. The codes are the values that result after the raw key codes produced by the hardware have been mapped through the key-map resource. See “Key Translation” on page C-8. Other keyboards can produce different virtual key codes; some produce raw key codes only.

The Apple Extended Keyboard may be connected to the Apple Desktop Bus (ADB) of any computer in the Macintosh II or Macintosh SE family. It contains duplicated Shift, Option, and Control keys to the right of the Space bar. Other keyboards have different physical layouts.

Figure C-1 Apple Keyboard II (domestic layout)

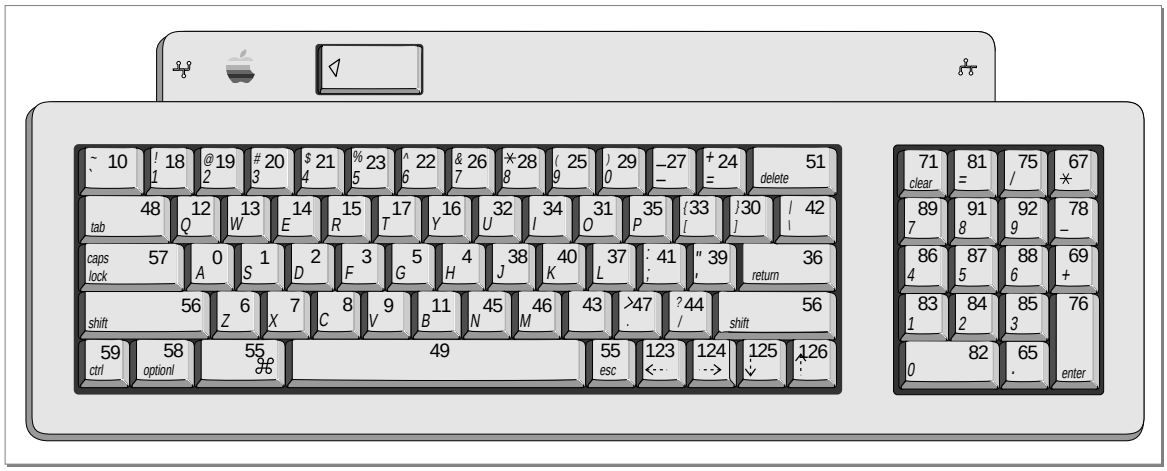


Figure C-2 Apple Extended Keyboard II (domestic layout)

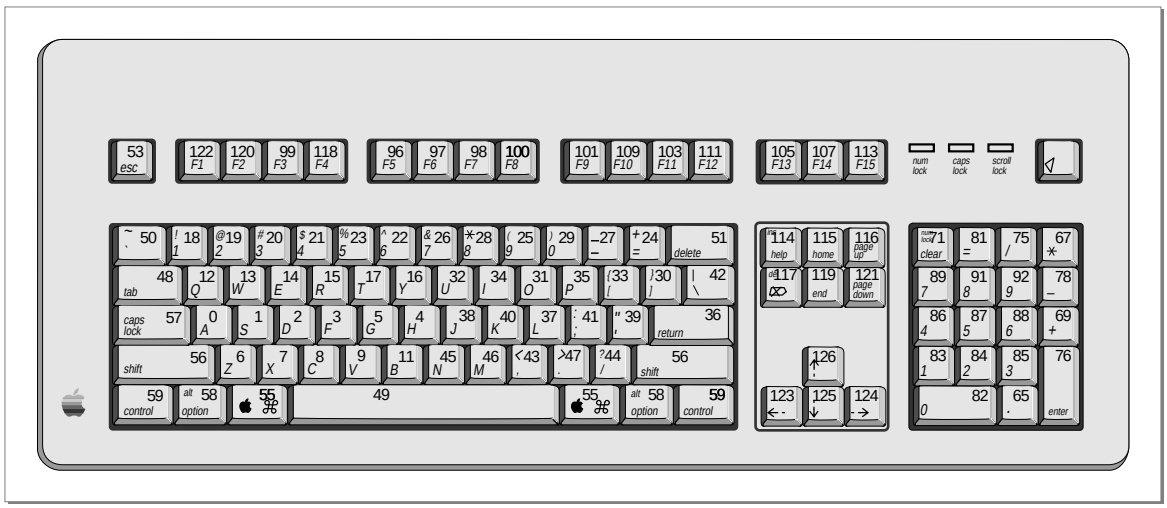


Table C-2 shows the keyboard modifier bits in the high byte of the `modifiers` field of an event record (defined by the `EventRecord` data type). The byte consisting of these bits is used to control the selection of tables in the keyboard-layout resource. See "Keyboard-Layout Resource (Type 'KCHR')" beginning on page C-18.

Table C-2 The keyboard modifier bits in an event record

Bit	Key
7	(Right Control if used)*
6	(Right Option if used)*
5	(Right Shift if used)*
4	Control (Left Control if different from Right Control)
3	Option (Left Option if different from Right Option)
2	Caps Lock
1	Shift (Left Shift if different from Right Shift)
0	Command

* See "Reassigning Right-Hand Key Codes" beginning on page C-14.

About the Keyboard Resources

The keyboard resources are Macintosh resources that facilitate worldwide keyboard handling and support the Macintosh script management system. They specify how keyboard input is converted to text for a particular writing system, language, or region. The Event Manager, the Script Manager, and the Menu Manager use the information in these resources to convert keystrokes to character codes, to switch input among different script systems, and to display the icon of the current keyboard in the Keyboard menu.

Note
Other Apple publications use the term *ASCII code* for **character code** (the 8-bit integer representing a text character generated by a key or a combination of keys on the keyboard or keypad) and the terms *key-down transition code* and *response code* for **virtual key code** (the key code that actually appears in keyboard events—that is, the value produced after a **raw key code** [the original value generated by a keyboard] has been mapped through the key-map resource). The terms *character code, raw key code,* and *virtual key code* are preferred in this book. *ASCII code* is limited here to the 7-bit code representing a character from the lower half of the Standard Roman character set. ◆

What the Keyboard Resources Are

The keyboard resources fall into two categories: those that are hardware-dependent (and script-independent) and those that are script-dependent (and hardware-independent). It is this division that allows many different physical keyboards to work correctly with many different script systems. Table C-3 lists the keyboard resources and their resource types, and gives a capsule description of their contents. More complete descriptions follow.

Table C-3 The keyboard resources

Name	Resource type	Contents
Key map	`'KMAP'`	Tables to map raw key codes to virtual key codes
Key remap	`'itlk'`	Tables to remap virtual key codes for certain key combinations
Keyboard layout	`'KCHR'`	Tables to map virtual key codes to character codes
Keyboard icons	`'kcs#'`	Keyboard icon (1-bit; black-and-white)
	`'kcs4'`	Keyboard icon (4-bit)
	`'kcs8'`	Keyboard icon (8-bit)
Keyboard swap	`'KSWP'`	Table specifying key combinations for changing keyboard script or input method
Key caps	`'KCAP'`	Data that determines keyboard display

- Key-map resource. Maps the raw key codes that have been generated by a specific keyboard microprocessor into hardware-independent standard virtual key codes. There is a maximum of one key-map resource per physical keyboard (several keyboards can share a single key-map resource).
- Key-remap resource. Remaps the virtual key codes for certain key combinations on certain keyboards to other virtual key codes, to allow a single keyboard-layout resource to work with all keyboards. This resource is optional; it is provided with certain keyboard-layout resources.
- Keyboard-layout resource. Maps virtual key codes to character codes. The keyboard-layout resource implements the character set for a script system. It is with different keyboard-layout resources that text input for different script systems and localized versions of system software is enabled. A script system has one or more keyboard-layout resources.

- Keyboard icon family. Implements keyboard icons—small icons that represent a keyboard script or input method—for screens of different bit depths (black-and white, 4-bit, and 8-bit, respectively). These icons are used in the Keyboard menu and in the Keyboard control panel. There is one icon family per keyboard-layout resource (or input method).
- Keyboard-swap resource. Lists modifier-plus-key combinations that can be used to change the keyboard script, or the keyboard layout or input method within a script. There is one keyboard-swap resource per version of system software.
- Key-caps resource. Specifies the physical arrangement of keys on a keyboard and is used to display the characters produced by each keypress. The key-caps resource is independent of any script system, but the Key Caps desk accessory uses it along with the keyboard-layout resource of the current script system—and a font in the current script system—to display the characters corresponding to each keypress or combination of keypresses.There is one key-caps resource per physical keyboard.

Keyboard resources and localized system software

When Macintosh system software is localized for a non-U.S. market, it contains replacements for or modifications to some of the U.S. versions of the keyboard resources. See the discussion of U.S. international resources and keyboard resources in the appendix "Built-in Script Support" for a list of resources that may be replaced during localization. ◆

Key Translation

Key translation is the conversion of keystrokes to character codes. In early versions of the Macintosh, keyboard translation was simple and direct: two low-memory pointers in the System file (accessed through global variables `Key1Trans` or `Key2Trans`) pointed to the translation routines. Those pointers are still available and are called by the Macintosh Plus but are not called by newer systems. The pointers are preserved so that applications that call them can still function correctly. However, they now point to a routine that implements a new standard mechanism.

The standard mechanism was developed with the advent of ADB keyboards; it was needed to map the different sets of raw key codes to a standard set of virtual key codes, which could in turn be mapped to character codes. In the standard method, a keystroke generates an interrupt; the keyboard driver maps the raw key code to a virtual key code, which it sends to the Event Manager; the Event Manager maps the virtual key code to a character code, and returns the character code to the driver. The driver in turn posts the key-down event. This method has two advantages:

- The mapping from raw key code to virtual key code achieves keyboard hardware independence. The raw mapping routine uses the table of a key-map resource for the keyboard, in the System file or in ROM.
- The mapping from virtual key code to character code allows support of multiple character sets. It is performed by the Event Manager `KeyTranslate` function, which is accessed through the `_KeyTrans` trap (not to be confused with the `Key1Trans` or `Key2Trans` pointers). `KeyTranslate` maps the virtual key code (plus modifiers, if any) to a character code, using tables in a keyboard-layout resource, also in the System file or in ROM. (`KeyTranslate` also handles *dead keys;* see page C-19.)

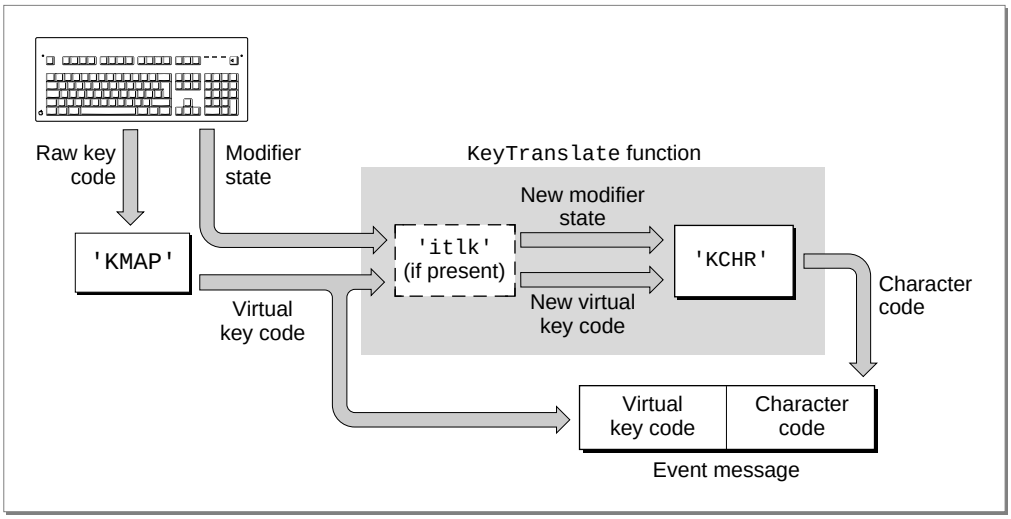
The Macintosh keyboard routines handle the keyboard properly for all script systems. Except for purely hardware-specific characteristics such as controlling lights on the keyboard, the function of the keyboard is completely determined by character-encoding tables in the keyboard-layout resource (with an optional associated key-remap resource). For each virtual key code and each possible modifier-key state, the character-encoding tables specify the equivalent character code. Figure C-3 summarizes the key translation process:

1. A keystroke initially produces a raw key code.
2. The keyboard driver uses the hardware-dependent key-map resource to map that raw key code into a hardware-independent virtual key code, and to set bits indicating the state of the modifier keys.
3. It then calls the Event Manager `KeyTranslate` function. The optional key-remap resource specifies how `KeyTranslate` should remap certain key combinations on certain keyboards before it performs its mapping. The key-remap resource reintroduces hardware dependence because certain scripts, languages, and regions need subtle differences in layout for specific keyboards. Generally, the key-remap resource affects only a few keys.
4. `KeyTranslate` uses the keyboard-layout resource to map a modifier state and a virtual key code into a character code, such as an ASCII code.
5. `KeyTranslate` returns the character code, and if the character code is nonzero the keyboard driver posts the key-down event into the event queue.

The net result of the process of key translation is a virtual key code and a character code in the `message` field of an event record, and modifier-key information in the `modifiers` field of the event record.

Note
On the Macintosh Plus, the event record contains raw key codes, not virtual key codes. However, except in the case of the small Macintosh 512K Keyboard with ISO layout, the Macintosh Plus raw key codes are identical to the virtual key codes that would have been produced. ◆

Figure C-3 The key translation process



Using the Keyboard Resources

The Operating System, along with the Script Manager and other Macintosh system software managers, uses information in the keyboard resources to convert keystrokes into character codes; to display keyboard icons; to change the current keyboard script, keyboard layout, or input method when the user enters Command-key combinations; and to properly display keyboard layout with the Key Caps desk accessory.

Most applications do not handle any of these tasks and therefore have no need for direct access to any of the keyboard resources. However, if you have the following special software needs related to text input, you can use the keyboard resources to help meet them:

- If your application needs to provide better international support for Command-key equivalents or a custom keyboard-layout resource, you can use the Event Manager `KeyTranslate` function to get the information you need from the appropriate keyboard-layout resource. See “Special Uses for the KeyTranslate Function” beginning on page C-22.

- If you are creating your own localized version of a script system and need to allow text input in that script system, you may need to create or modify a keyboard-layout resource, and possible a key-remap resource. If you do make a new keyboard-layout resource, you also need to create a keyboard icon family to accompany it. To do that you will need the information in "Keyboard-Layout Resource (Type 'KCHR')" beginning on page C-18, "Key-Remap Resource (Type 'itlk')" beginning on page C-16, and "Keyboard Icon Family (Types 'kcs#', 'kcs4', 'kcs8')" beginning on page C-25.
- If you are designing a new type of keyboard, you need to make sure it produces the appropriate raw key codes. See the next section. Each new keyboard also needs to work correctly with the Key Caps desk accessory; see "Key-Caps Resource (Type 'KCAP')" beginning on page C-28. Note that hardware development is beyond the scope of *Inside Macintosh.* See *Guide to the Macintosh Family Hardware* and contact Macintosh Developer Technical Support for more information.

Key-Map Resource (Type 'KMAP')

The key-map resource (resource type `'KMAP'`) is used for converting the raw key codes produced by a keyboard's microprocessor into hardware-independent virtual key codes. There is one key-map resource per physical keyboard on a Macintosh; it belongs to the Operating System, not to any script system.

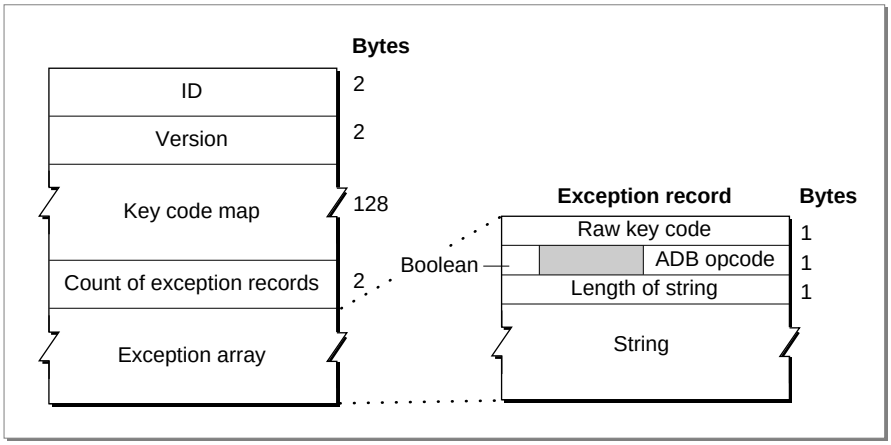
The key-map resource ID number equals the ID number of the type of keyboard it is associated with. See Table C-1 on page C-4. If a matching key-map resource cannot be found for the keyboard in use, the Operating System substitutes the `'KMAP'` resource whose ID is 0; on all Macintosh systems later than the Macintosh Plus, the key-map resource with ID = 0 is in ROM.

Note
Most current keyboards use the key-map resource with ID = 0. However, keyboard types 2 and 5, for example, require their own key-map resources. ◆

The key-map resource contains a 128-byte table that provides a one-to-one mapping of raw key codes to virtual key codes—the first byte contains the virtual key code for a raw key code of $00, the second for $01, and so forth. The table is followed by an array of exceptions. The high bit of the byte containing the virtual key code signals an exception entry in the exception array. (Virtual key codes themselves are only 7 bits long.)

The exception array lets the device driver initiate communication with the device, usually to perform a state change—for example, to send codes to the keyboard that instruct it to turn on lights when a given key such as Caps Lock is down. The exception array begins with a 2-byte record count followed by that many records. The format of the key-map resource and its exception array is shown in Figure C-4.

Figure C-4 Format of the key-map resource



The elements in the resource have these meanings:

- ID. The resource ID for this particular key-map resource.
- Version. The version number of this key-map resource format.
- Key code map. A 128-byte table that contains virtual key codes. At each byte offset into the table, the entry is the virtual key code (plus possibly an exception entry flag) for the raw key code whose value equals that offset.
- Count of exception records. The number of entries in the exception array.
- Exception array. An array of exception records, which map raw key codes to communication instructions.

Each exception record has these elements (see also Figure C-4):

- A raw key code.
- One byte containing the following elements:
  - A Boolean (`Xor` or `noXor`) field that determines whether to instruct the driver to invert the state of the key instead of using the state provided by the hardware.
  - Filler (3 bits in length).
  - The ADB opcode, an instruction to the keyboard to perform some task. ADB opcodes are described in *Inside Macintosh: Devices.*

- A variable length Pascal data string that is passed to the ADB op trap along with the ADB opcode. The first byte in the string is the length byte.

The following is an example of the exception array used to turn the Caps Lock light of the Apple Extended Keyboard II on and off, to match the state of the Caps Lock key.

```
{
        $39, noXor, $E, "\$00\$02";
        $B9, noXor, $E, "\$00\$02";
}
```

Note
Do not change the key-map resource. Everything your application needs to support any kind of text input is in the keyboard-layout and key-remap resources. You need to work with the key-map resource only if you are making your own keyboard. ◆

Apple Extended Keyboard

With the Apple Extended Keyboard (and Apple Extended Keyboard II, shown in Figure C-2), the standard key-map resource that is supplied with the system converts the following raw key codes to virtual key codes, as listed in Table C-4.

Table C-4 Key-map resource assignment of raw key codes to virtual key codes

Key	Raw key code	Virtual key code
Control	$36	$3B
Left Arrow	$3B	$7B
Right Arrow	$3C	$7C
Down Arrow	$3D	$7D
Up Arrow	$3E	$7E

The standard key-map resource leaves all other virtual key codes identical to the raw key codes they are generated from.

Reassigning Right-Hand Key Codes

It is possible to reassign the standard raw key codes and virtual key codes for the Shift, Option, and Control keys on the right side of the Apple Extended Keyboard, in order to distinguish right-side keystrokes from left-side keystrokes for those keys. To do so, you need to obtain the special values listed in Table C-5.

Table C-5 Reassigning right key codes for Shift, Option, and Control keys

Right key	Normal raw	Normal virtual	Special raw	Special virtual
Shift	$38	$38	$7B	$3C
Option	$3A	$3A	$7C	$3D
Control	$36	$3B	$7D	$3E

The normal raw and virtual key codes for Right-Shift, Right-Option, and Right-Control keys correspond to the left versions of these keys. You can obtain the special raw and virtual key codes only by changing the value of the device handler ID field in the Apple Extended Keyboard's register 3 from 2 to 3. For details about the device handler ID field, see *Inside Macintosh: Devices*.

▲ **WARNING**
This capability is included for compatibility with certain existing operating systems that distinguish between the left and right versions of these keys. Its use by new applications violates the Apple human interface guidelines and is strongly discouraged. ▲

Other Hardware Dependencies

The principle underlying virtual key codes is to have a single unique code per character code, regardless of the keyboard used. Nevertheless, some hardware dependencies remain:

- The small Macintosh 512K Keyboard with ISO layout and the ISO ADB keyboards have an extra key not present on domestic keyboards. This key produces a virtual key code of $0A.

- There is a different virtual key code for the Enter key, depending on whether it is on the keypad ($4C on the Macintosh Plus keyboard and most ADB keyboards), or on the main section of the keyboard ($34 on the original Macintosh keyboard and the Macintosh Portable and PowerBook keyboards).
- Virtual key codes for cursor keys and some keypad operator keys differ between ADB keyboards and non-ADB (Macintosh Plus) keyboards, as shown in Table C-6. Note that on Macintosh Plus keyboards, the virtual key codes for keypad operators are the same as the virtual key codes for cursor keys. The Shift modifier controls which character code is generated. On these keyboards, for example, holding down the Shift key and pressing the Left Arrow key produces the plus character (+).

Table C-6 ADB and non-ADB virtual key codes for cursor keys and keypad keys

Key	ADB code	Non-ADB code (Macintosh Plus)
Left Arrow	$7B	$46
Right Arrow	$7C	$42
Down Arrow	$7D	$48
Up Arrow	$7E	$4D
Keypad Plus (+)	$45	$46 (with Shift bit set in modifiers)
Keypad Asterisk (*)	$43	$42 (with Shift bit set in modifiers)
Keypad Equal (=)	$51	$48 (with Shift bit set in modifiers)
Keypad Slash (/)	$4B	$4D (with Shift bit set in modifiers)

Virtual Key Codes for Non-ADB Keyboards

The original Macintosh keyboard (for both the 128K and 512K versions) and the Macintosh Plus keyboard produce event records with raw key codes rather than virtual key codes, because there is no key-map resource for them. For domestic versions of these keyboards it is not a problem, because the raw key codes are identical to the virtual key codes expected by the U.S. keyboard-layout resource. The international version of the Macintosh Plus keyboard, however, and the ISO layout of the small Macintosh 512K keyboard, produce raw key codes that cannot be treated as virtual.

When a keypress from the international version of the Macintosh Plus keyboard occurs, the interrupt handler calls the `_Key1Trans` hook, which translates the raw key codes to virtual key codes before calling `KeyTranslate`. Thus your application normally receives the correct character codes even if an international version of the Macintosh Plus keyboard is attached. However, the raw key code is what is placed in the event record. Therefore, if you need to explicitly convert raw key codes to virtual key codes, you can use the values in Table C-7. Raw key codes are offsets into the table; the byte at each offset represents the virtual key code for that raw key code. (The keyboard produces raw key codes up to $3F only; key codes above that value are generated by an optional keypad.)

Table C-7 Virtual key codes for the international Macintosh Plus keyboard

Raw codes	Virtual codes							
$00–$07	$00	$01	$02	$03	$04	$05	$32	$06
$08–$0F	$07	$08	$2C	$09	$0C	$0D	$0E	$0F
$10–$17	$10	$11	$12	$13	$14	$15	$16	$17
$18–$1F	$18	$19	$1A	$1B	$1C	$1D	$1E	$1F
$20–$27	$20	$21	$22	$23	$2A	$25	$26	$27
$28–$2F	$28	$29	$24	$2E	$2F	$0B	$2D	$2B
$30–$37	$30	$34	$0A	$33	$31	$35	$36	$37
$38–$3F	$38	$39	$3A	$3B	$3C	$3D	$3E	$3F

The domestic and ISO layouts of the small Macintosh 512K keyboard have keyboard types of 3 and 259, respectively. However, in both cases the low-memory global that specifies current keyboard type (`KbdType`) holds the value 3. The user indicates which keyboard is in use through a control in the Keyboard control panel that appears only on non-ADB systems. The user's selection is kept in the `itlcOldKeyboard` field of the system script's international configuration (`'itlc'`) resource. You can examine that field if you need to know whether the ISO or domestic layout of the small Macintosh 512K keyboard is in use.

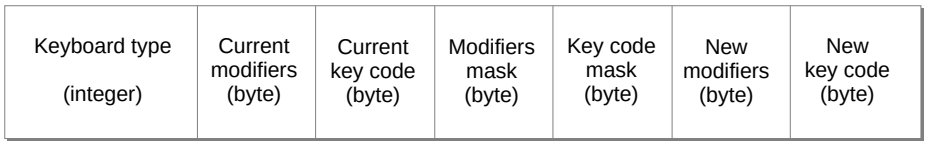
Key-Remap Resource (Type 'itlk')

The key-remap resource (resource type `'itlk'`) is used by the `KeyTranslate` function to ensure that all international keyboard layouts work on all Macintosh keyboards. The key-remap resource specifies how to remap the virtual key codes produced by certain key combinations before `KeyTranslate` converts the virtual key codes to character codes with a keyboard-layout (`'KCHR'`) resource. `KeyTranslate` is described in the chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials.*

There is one key-remap resource per keyboard-layout resource that needs it. The `'itlk'` resource has the same resource ID as the keyboard-layout resource with which it is associated. The Operating System loads key-remap resources from the System file only.

The key-remap resource consists of an integer count of entries followed by a set of 8-byte entries. Figure C-5 shows the format of an entry.

Figure C-5 Format of an entry in the key-remap resource



Before the `KeyTranslate` function begins processing with the keyboard-layout resource, it determines which entry in the key-remap resource to use. It tests each entry in the key-remap resource to see whether

- the actual keyboard type matches the keyboard type element
- the product of an `AND` operation on the actual virtual key code with the key code mask matches the current key code element
- the product of an `AND` operation on the actual modifiers with the modifiers mask matches the current modifiers element

If all three match, `KeyTranslate` substitutes the new modifiers and virtual key code from that entry before applying them to the keyboard-layout resource.

To allow for a more compact table when several virtual key codes produced from one key (using different modifiers) are all mapped together to a different key, an additional step is taken. `KeyTranslate` uses the modifiers mask and key-code mask in the key-remap entry to produce a number of new modifiers and virtual key codes. Here is how a single entry can remap all modifier combinations for a given key:

1. An `AND` operation is performed on the new modifiers and new virtual key code with the modifiers mask and the key-code mask from the entry.
2. An `AND` operation is performed on the actual modifiers and actual virtual key code with the 1's complement of the modifiers mask and key code mask from the entry.
3. The `OR` of these two operations is the final result that is used for key translation.

Note
If the keyboard type is 259 (the ID for the ISO layout of the small Macintosh 512K Keyboard), the third field in the key-remap resource (which usually contains the current virtual key code) consists of the raw key code. See Table C-1 on page C-4 of this appendix for a list of the keyboard types. ◆

Keyboard-Layout Resource (Type 'KCHR')

The keyboard-layout resource (resource type `'KCHR'`) specifies the mapping of virtual key codes to character codes. Each installed script system has one or more keyboard-layout resources; there may be one or more for each language or region to suit the preference of the user. The resource ID for each keyboard-layout resource is within the range of resource ID numbers for its script system. The ID number of the default `'KCHR'` resource for a script system is specified in the `itlbKeys` field of the script's international bundle (`'itlb'`) resource.

U.S. keyboard-layout resource

Specific features of the U.S. keyboard-layout resource (ID = 0) are described in the appendix "Built-in Script Support" in this book. ◆

Keyboard-layout resources for 2-byte script systems

Keyboard-layout resources for 2-byte script systems have the same size and function as those for 1-byte script systems; they generate 1-byte character codes only. It is the input method that is responsible for producing the final 1-byte or 2-byte character codes. ◆

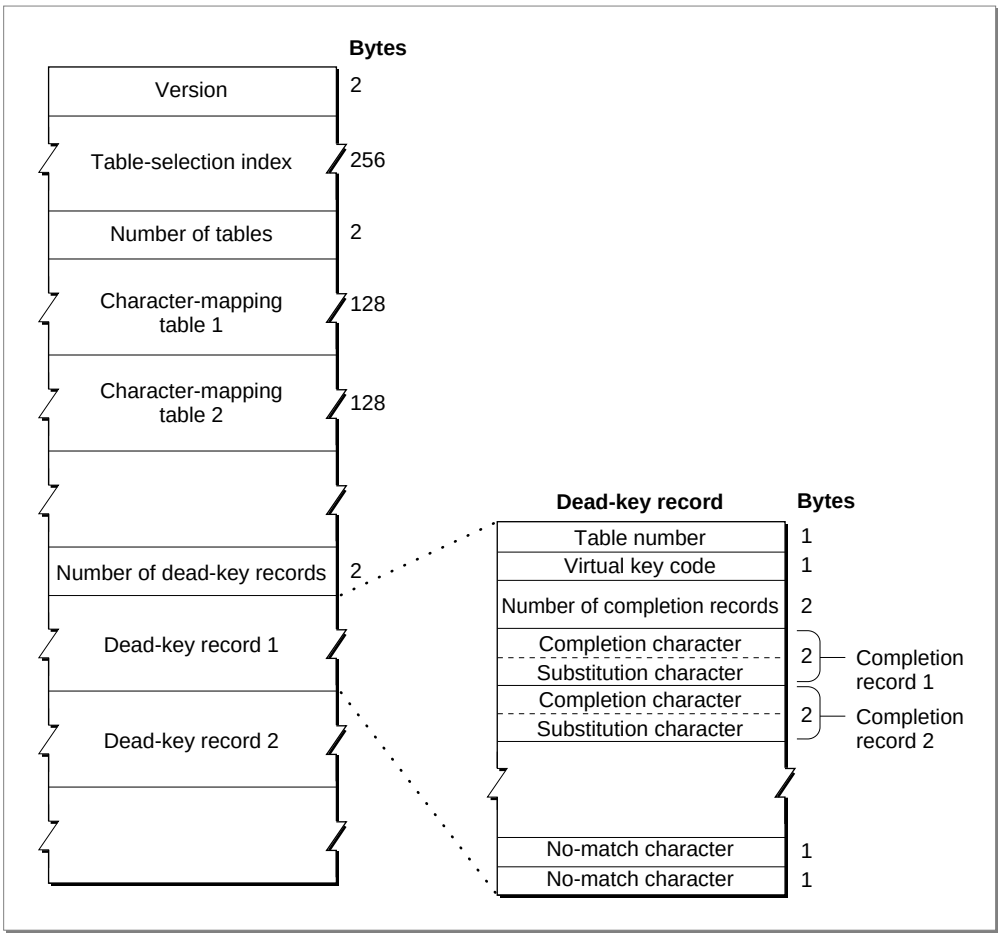
Resource Format

Figure C-6 shows the format of the keyboard-layout resource. Its header consists of a version number only. The header is followed by a 256-byte table-selection index that is used to access character-mapping tables. The index is followed by the character-mapping tables, a series of 128-byte tables that map virtual key codes to character codes, depending on what modifier keys are pressed. The final part of the resource is a dead-key table, a series of records that define dead keys and completers. The dead-key records allow the user to enter special character forms, such as accented characters, from the keyboard. How dead keys are processsed is described under "The KeyTranslate Function and the Keyboard-Layout Resource" beginning on page C-19.

The dead-key table consists of a 2-byte count of dead-key records, followed by that many records. A dead key record consists of a 1-byte table number (corresponding to a character-mapping table), a 1-byte virtual key code (without up/down bit), a completion table, and a no-match character.

Each completion table in a dead-key record consists of a count of completion records, followed by that number of completion records. A completion record is simply a substitution pair for character codes. If the character code matches the first byte in the completion record, the second byte is substituted for it.

Figure C-6 Format of the keyboard-layout resource



The KeyTranslate Function and the Keyboard-Layout Resource

During the process of key translation, the Event Manager `KeyTranslate` function applies the virtual key code and the state of the modifier keys to the keyboard-layout resource to determine a character code. Table C-2 on page C-6 shows the meanings of the keyboard modifier bits in the high-order byte of the `modifiers` field of an event record (defined by the `EventRecord` data type). The `KeyTranslate` function uses the byte value determined by the settings of these bits to control the selection of tables in the keyboard-layout resource.

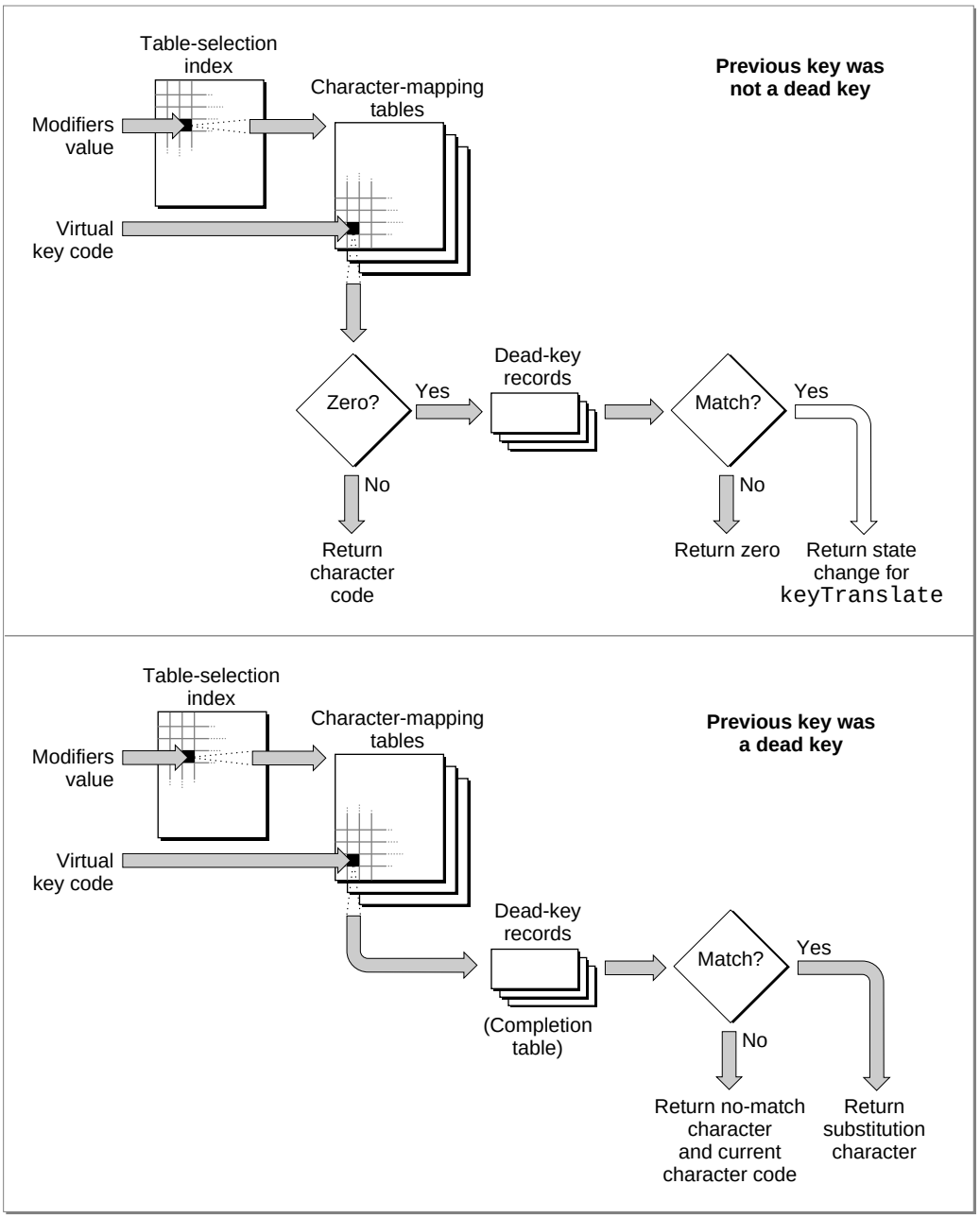
Figure C-7 gives an overview of how the parts of the keyboard-layout resource are used. It starts when the user presses a key or combination of keys, and the Event Manager passes the virtual key code and the state of the modifier keys to the `KeyTranslate` function:

1. First, `KeyTranslate` treats the modifier state information—8 bits, each bit indicating the state of one modifier key—as a byte whose value is used as an index into the 256-byte table-selection index to get a table code. The table code specifies which of the 128-byte character-mapping tables to use to map the virtual key code to a character code.
2. `KeyTranslate` uses the virtual key code as an index into the selected character-mapping table. If the table has a nonzero entry for the virtual key code, that entry is the desired character code. `KeyTranslate` returns that character code and the Event Manager posts a key-down event—*unless* the previous keypress had been a dead key. See step 4.
3. If the entry in the character-mapping table is 0, `KeyTranslate` searches the dead-key table. It looks for a match with both the virtual key code and the table number fields in a dead-key record. If there is no match, `KeyTranslate` returns 0. If there is a match,the dead-key information is preserved in the `state` parameter of the `KeyTranslate` function. `KeyTranslate` returns 0, so no event is posted, but the state information affects how the next virtual key code is to be processed.
4. If the previous key was a dead key, `KeyTranslate` searches the completion table in the dead-key record corresponding to the previous keypress. If the character code of the current keypress matches the first byte of any completion record in the completion table, the second byte in the record is substituted for it. If it does not match any first bytes in the completion table, the current character code is preceded by the no-match character found at the end of the dead key record and `KeyTranslate` returns both characters.

   For instance, in the U.S. keyboard-layout resource the Option-E combination is a dead key. When pressed, no character appears on the screen, but the `state` parameter of `KeyTranslate` is modified to hold the information that the dead key for the acute accent (´) has been pressed. If the next character is a valid completer key (such as a, e, i, o, or u), `KeyTranslate` returns the equivalent substitution character (á, é, í, ó, ú), an event is posted, and the character appears on the screen. If the next character is not a valid completer (for example, x), `KeyTranslate` returns *both* the no-match character (typically the accent character by itself) and the current character code; two events are posted, and both characters appear on the screen (´x).

As far as your application is concerned, no event is generated by pressing a dead key. The only information you receive regarding the dead key is after the fact. When the user produces “Á” by pressing Option-E followed by “A”, you receive a single event containing a virtual key code corresponding to “A”, no modifiers, and a character code of “Á”.

Figure C-7 Inside the keyboard-layout resource



Special Uses for the KeyTranslate Function

In normal key translation, the Event Manager `KeyTranslate` function performs the conversion from virtual key code to character code and passes the result to your application in the `message` field of the event record for a key-down event. The script management system provides `KeyTranslate` with a pointer to the proper keyboard-layout resource to use, based on the current script.

There may be situations, however, in which you may want to explicitly call `KeyTranslate`, either to install your own keyboard layout or to perform special processing.

Installing a Custom Keyboard-Layout Resource

The script management system loads and uses only those keyboard-layout resources that are installed in the System file. It cannot load a keyboard-layout resource that is, for example, in the resource fork of your application. However, if your application needs to modify the keyboard layout temporarily without forcing users to install a new keyboard layout, you can load a keyboard-layout resource from your own application resource fork and call `KeyTranslate` directly after each key-down event, passing it a pointer to that keyboard-layout resource and using the same virtual key code and modifiers that you received in the event message.

To more permanently replace a script system's keyboard layout, you can have the user install a keyboard-layout resource and a keyboard-icon family in the System file. Both resources must have identical ID numbers, in the range for the script system for which they will be used. You call the Script Manager `SetScriptVariable` procedure twice, to make those IDs the defaults for the given script system. You then call the Script Manager `KeyScript` procedure to load the resources and make them available to the system. Listing C-1 demonstrates the calls for a Dvorak keyboard layout in the Roman script system.

Listing C-1 Loading a non-system keyboard-layout resource

```
CONST
   DvorakID =  500;

VAR
   err:  OSErr;

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

In this example you do not need to call `KeyTranslate` to get character codes for the new keyboard layout, and the new keyboard layout will be in effect until the system is restarted or until your application restores the original keyboard layout.

The most permanent way to replace the keyboard layout is to make the system use your keyboard layout as its default. To do that you must modify the `itlbKeys` field of the target script system's international bundle (`'itlb'`) resource. The international bundle resource is described in the appendix "International Resources" in this book.

IMPORTANT
Apple Computer's system software licensing policy forbids shipping a modified System file. If you want to modify the System file, it is best to have the user either run the Installer to install your resources, or drag a file consisting of those resources onto the System Folder. Contact Macintosh Developer Technical Support for information on the Installer. ▲

You can inspect and edit any keyboard-layout resource by using a resource editor such as ResEdit.

Using KeyTranslate for Command-Key Equivalents

In some cases you may need to call `KeyTranslate` to regenerate a different character code using the same keyboard-layout resource. For example, the U.S. `'KCHR'` and some other Roman keyboard layouts ignore the Shift modifier key if the Command modifier key is also pressed. That means you cannot directly use uppercase characters or shifted symbols as Command equivalents. Furthermore, for those keyboard layouts where the period is a shifted key, it means that the standard Macintosh command to cancel an operation (Command-period) cannot be generated. As another example, some applications that accept Command-? as a request for Help simply assume that "?" is a shifted version of "/", and thus bring up a Help window whenever the Command key and "/" are pressed simultaneously. This gives incorrect behavior on keyboards in which "?" is not generated by Shift-/.

To overcome this and similar difficulties, you can use the virtual key code you receive in the key-down event record, and call `KeyTranslate` to run it back through the same keyboard-layout resource, but without the modifier(s) that applied when the character code was first generated. If the resulting character code is one that is significant for a command equivalent, you can use it plus the modifier state that originally applied to decide what action to take.

Listing C-2 is a routine that removes the Command-key bit from the modifiers field of an event record and runs the same virtual key code through `KeyTranslate`, using the same keyboard-layout resource, to see if a different character code results.

Listing C-2 Regenerating a character code with `KeyTranslate`

```
FUNCTION TryAgain(myEvent: EventRecord): LongInt;

CONST
   newModifierMask = $FE00;            {turn off cmdKey bit}

VAR
   Modifiers:     Integer;
   VirtualCode:   Integer;
   KeyCode:       Integer;
   someState:     LongInt;
   KCHRPtr:       Ptr;

BEGIN
                                       {don't keep cmdKey bit}
   Modifiers := BAnd(myEvent.modifiers, newModifierMask);
                                       {keep virtual key code, put in low byte}
   VirtualCode := BSR(BAnd(myEvent.message, keyCodeMask), 8);
                                       {assemble new key code for KeyTranslate}
   KeyCode := BOr(Modifiers, VirtualCode);
                                       {get pointer to current 'KCHR'}
   KCHRPtr := Ptr(GetScriptManagerMVariable(smKCHRCache));
   someState := 0;                     {initialize KeyTranslate dead-key state}
                                       {see what ascii code is returned}
   TryAgain := KeyTranslate(KCHRPtr, KeyCode, someState);
                                       {look for returned values in both }
                                       { high and low word of result}
END;
```

In designing Command equivalents for your application, keep in mind that there may be less chance of inconsistency and confusion if you present Command equivalents to the user—and interpret them yourself—as grouped modifiers applied to the basic (unshifted) character you want to use for the command. (Note, however, that to do so you would have to write your own custom menu-definition resource.) For example, you might show "Command-Option-P" in the menu rather than "Command-π"; when interpreting it, you could use `KeyTranslate` and the virtual key code in the event record to make sure that the key for "p" was pressed, rather than just assuming that "π" is produced by Option-P.

Another possibility is to define few Commmand-key equivalents yourself, and to let the user create as many equivalents as desired.

Keyboard Icon Family (Types 'kcs#', 'kcs4', 'kcs8')

The keyboard icon family is a set of resources (resource types `'kcs#'`, `'kcs4'`, and `'kcs8'`) that specify a family of small icons representing a keyboard layout. They define black-and-white, 4-bit, and 8-bit small color icons, respectively. There is one keyboard icon family per keyboard-layout resource; each of the keyboard icon resources has the same resource ID as the keyboard-layout resource with which it is associated.

The Operating System loads keyboard icon resources from the System file only. The ID number of the default keyboard icon family for a script system is specified in the `itlbIcon` field of the script's international bundle (`'itlb'`) resource. However, the Operating System ignores this value and instead looks for a keyboard icon family whose resource ID matches the ID of the keyboard-layout resource it is loading. If it cannot find an icon family with that ID, the Operating System loads the default keyboard icon suite (ID = –16491).

Some differences exist between the keyboard icon family and the color icon families used elsewhere in the Macintosh Operating System. First, only small icons (16-by-16 pixels) are supplied; there are no large keyboard icons (32-by-32 pixels). Second, the resource type for keyboard small color icons is different from the resource type used elsewhere for small color icons (`'ics#'`, `'ics4'`, and `'ics8'`). This difference is to avoid resource ID conflicts with those icon resources, because the keyboard color icons may have IDs anywhere in the range 0–32767, and certain negative ranges as well. The keyboard icon types and the equivalent standard color icon types are shown in Table C-8.

Table C-8 Keyboard color icon types and standard icon equivalents

Keyboard icon type	Standard icon equivalent	Bit depth
`'kcs#'`	`'ics#'`	1
`'kcs4'`	`'ics4'`	4
`'kcs8'`	`'ics8'`	8

Note
If the 4-bit and 8-bit icons (resources `'kcs4'` and `'kcs8'`) in your application have exactly the same appearance and colors, then you only need to provide a 4-bit icon. ◆

The keyboard icons are used in the Keyboard control panel and in the Keyboard menu when it is displayed. In Macintosh system software versions 7.0 and later, the Keyboard menu always appears when more than one script system is enabled, and may be forced to appear even if only one script system is present (if the `smfShowIcon` flag in the Script Manager general flags is set at startup).

Figure C-8 Sample keyboard icons

See the Finder Interface chapter of *Inside Macintosh: Macintosh Toolbox Essentials* for additional information on color icons and icon families. See also *Macintosh Human Interface Guidelines* for design suggestions for color icon families.

Keyboard-Swap Resource (Type 'KSWP')

The keyboard-swap resource (resource type `'KSWP'`) specifies the modifier-plus-key combinations with which the user can change keyboard scripts, keyboard layouts within scripts, and input methods. For example, the standard keyboard-swap resource specifies that pressing Command–Space bar changes the keyboard to the default keyboard for the next script. (In this case, *next* means next in the Keyboard menu.)

There is one keyboard-swap resource per localized version of system software. A localized system may either use the standard `'KSWP'` resource or replace it with one of its own. The keyboard-swap resource is in the System file; its resource ID is 0.

The keyboard-swap resource consists of an array with series of entries, each of which specifies modifier-plus-key combinations that can be used to change keyboard layouts and scripts. Figure C-9 shows the format of entries in the `'KSWP'` resource.

Figure C-9 Format of entries in the keyboard-swap resource

Script code or special negative code (integer)	Virtual key code (byte)	Modifier state (byte)

The elements of the entry have these meanings:

- Script code or negative code. The code number of a script system—such as 0 (`smRoman`)—or a special negative code for switching. The special negative codes are identical to the selectors for the Script Manager `KeyScript` procedure. The selectors are listed and described along with the `KeyScript` procedure in the chapter "Script Manager" in this book.
- Virtual key code. The virtual key code (for example, $31 for Space bar) required to generate the script code or special negative code of this element.
- Modifier state. The modifier-key setting (for example, Command key down) that must accompany the virtual key code.

Listing C-3 is a Rez-format definition of a hypothetical keyboard-swap resource.

Listing C-3 A hypothetical keyboard-swap resource

```
resource 'KSWP' (0, sysheap) {
      {/* array: 2 elements */
         /* [1] = smKeyNextScript */
         -1, $31, controlOff, optionOff, shiftOff, commandOn,
         /* [2] = smKeyNextKybd */
         -4, $31, controlOff, optionOn, shiftOff, commandOn,
      }
};
```

This resource defines a rotation to the next script system on Command–Space bar, and a rotation to the next keyboard layout on Command–Option–Space bar.

Note
The expression that evaluates the size of a keyboard-swap resource is complicated. If you need to perform a DeRez operation on a keyboard-swap resource, contact Macintosh Developer Technical Support for details. ◆

IMPORTANT
The Script Manager removes from the event queue any Command-key combinations involving the Space bar if that Command-key combination indicates a feature supported by the current script system. For example, if multiple script systems are installed, the Script Manager strips the Command–Space bar combination (which specifies changing script systems) from the event queue. If multiple script systems are not installed, this event is not removed, so users can use it in Command-key macros. Applications, however, should never depend on Command-key combinations involving the Space bar. ▲

Key-Caps Resource (Type 'KCAP')

The key-caps resource (resource type `'KCAP'`) reflects the physical layout of a particular keyboard and is used by the Key Caps desk accessory. The resource indicates the shapes and positions of all keys, and defines the virtual key codes that correspond to each physical key. Key Caps uses this resource to draw a representation of the current keyboard layout—using the current keyboard-layout resource—for the current physical keyboard. If you are creating a new keyboard, you can define its physical layout in a key-caps resource.

For system software versions 7.0 and later, the key-caps resource is located in the System file. There is one `'KCAP'` resource per physical keyboard on a Macintosh; it belongs to the Operating System, and not to any script system. The resource ID for each key-caps resource is equal to the keyboard type of the keyboard it is associated with. See Table C-1 on page C-4 for a list of keyboard types. For ADB keyboards, the ID of the key-caps resource is the same as the keyboard handler ID.

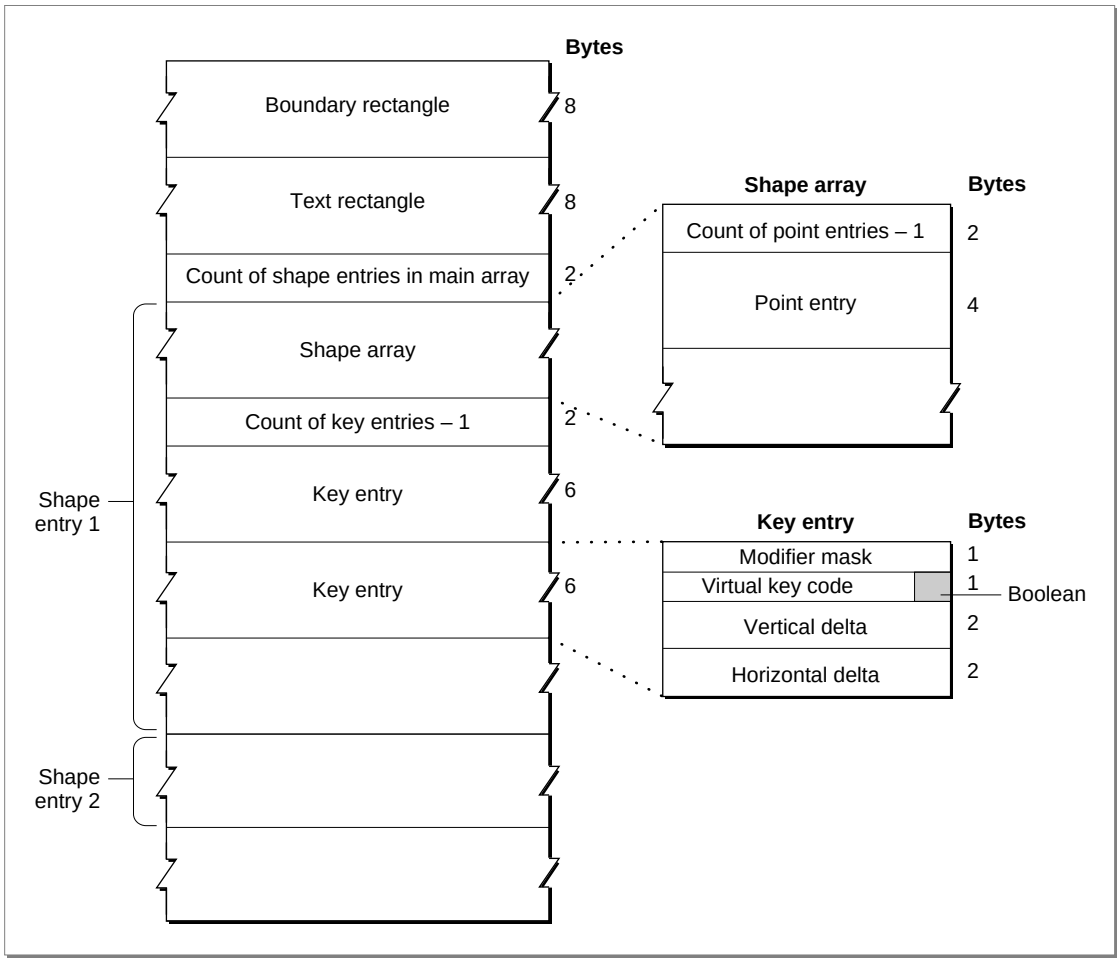
IMPORTANT

The key-caps resource should never require localization. The only time a key-caps resource needs to be added is for a keyboard that has a new physical arrangement (or a new keyboard handler ID). ▲

Resource Format

Figure C-10 shows the format for the key-caps resource.

Figure C-10 Format of the key-caps resource



The key-caps resource has these elements:

- Boundary rectangle. The position of the content region of the Key Caps window.
- Text rectangle. The position of the text box within the Key Caps window.
- Main array. The remainder of the resource. It consists of an array of one shape entry for each key shape.

Each shape entry in the main array has two components:

- Shape array. A (zero-based) count of entries followed by one or more entries. Each entry is a point, representing the relative pixel offset from the origin of the key, that define a particular key shape. The shape array is a single point for rectangular keys. More complex keys, like the Return key, need two points in their shape array.
- Key array. A set of key entries, describing all the keys with that shape.

Each key entry in a shape entry specifies the following information:

- Vertical delta and horizontal delta. Vertical and horizontal values to move the pen before drawing the current key. For each shape (that is, for each shape entry in the main array), the pen starts out at the upper-left corner of the content region of the Key Caps window, so the vertical and horizontal delta values for the first key in the key array for that shape are distances from the upper-left corner to the origin of the first key. For subsequent keys in the key array, the deltas are distances from the origin of the previous key to the origin of the current key. Each key is drawn with the shape defined by the shape array for that shape.
- Virtual key code. The virtual key code for the current key. Because it uses virtual key codes, each key-caps resource is tied directly to a particular key-map resource and hardware keyboard but can work with any keyboard-layout resource.
- Modifier mask and Boolean. A modifier mask and a Boolean flag for how to use it. When Key Caps draws the current key, it retrieves the byte that represents the real modifier key state, combines it with this mask performing an `OR` or `AND` operation as specified, calls the `KeyTranslate` function with the resulting modifier byte and the virtual key code from the key-caps resource, and draws the resulting character or characters in the current key's location. The modifier mask is only required for non-ADB keyboards, which use artificial modifier key states to overlap the key codes for arrow keys and keypad operator keys. For other keyboards, the mask is 0 and the flag is set to specify an `OR` operation.

Listing C-4 is an abridged example of the data in a key-caps resource, shown in Rez format.

Listing C-4 Sample key-caps resource data in Rez format

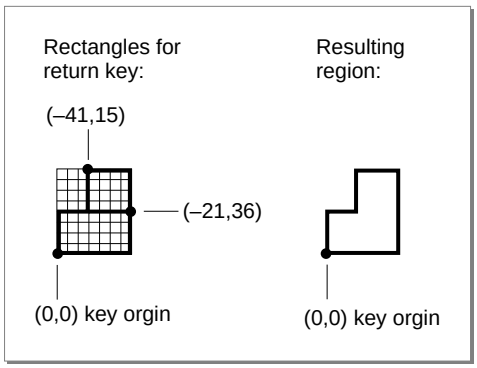
```
resource 'KCAP' ($01) {
            {60, 45, 220, 455},              /* boundsRect */
            {12, 42, 36, 368},               /* textRect */
            {
               { {21, 21} }, {               /* Shape No. 1 */
                  0, or, $35, 50, 10;           /* escape */
                  0, or, $12, 0, 20;            /* 1 ! */
                  0, or, $13, 0, 20;            /* 2 @ */
                  …
                  0, or, $7D, 0, 20;            /* Down arrow */
                  0, or, $7E, 0, 20;            /* Up arrow */
                  0, or, $41, 0, 80;            /* Keypad . */
                  0, or, $55, -20, 0;           /* Keypad 3 */
                  …
            };
               { {21, 31} }, {               /* Shape No. 2 */
                  0, or, $30, 70, 10;           /* Tab */
                  0, or, $33, -20, 260          /* Backspace */
            };
                  …
               {  {-21, 36}; {-41, 15} }, {  /* Shape No. 3 */
               0, or, $24, 111, 265             /* Return */
            };
            …
         }
};
```

The basic square key has a shape array of `{ {21, 21} }`, which puts the origin in the upper-left corner of the key. The first key in the key array for this shape is the Escape key (key code $35) in the upper-left corner of the keyboard; this key is at vertical and horizontal delta offsets of (50, 50) from the upper-left corner of the window's content region. The next key with this shape is immediately to the right, with its origin at delta offsets of (0, 20) from the origin of the previous key.

The next shape is the slightly wider key with a shape array of `{ {21, 31} }`, used for the Tab and Backspace keys. The origin of the Tab key is at offsets (70, 10) from the upper-left corner of the window's content region (which puts the Tab key one row below the Escape key).

The shape array for the Return key is `{ {-21, 36}; {-41, 15} }`, which means that it is the union of two rectangles: the first rectangle is from the origin of the key to the first point, and the second rectangle is from the first point to the second point. (Both points are measured relative to the key origin, however.) This shape array puts the Return key's origin in the lower-left corner of the key. See Figure C-11. The origin is at offsets (111, 265) from the upper-left corner of the window's content region.

Figure C-11 Shape array and resulting region for the Return key

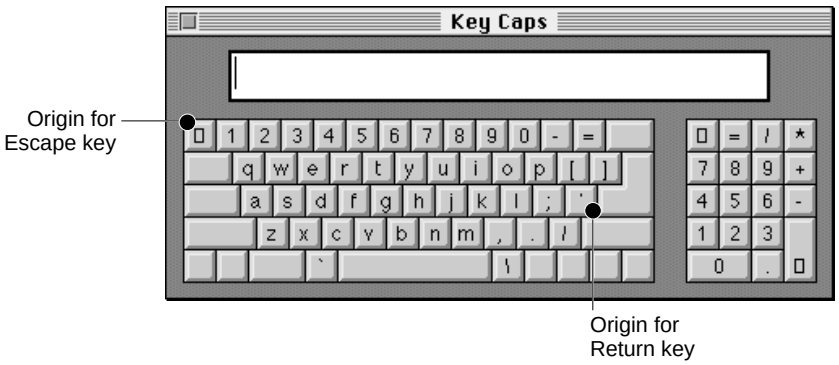


Key Caps Desk Accessory

This section discusses how the Key Caps desk accessory uses information in its key-caps resource to represent the physical layout of a keyboard. It also describes how the Key Caps desk accessory provides feedback to the user on how dead keys produce accented characters.

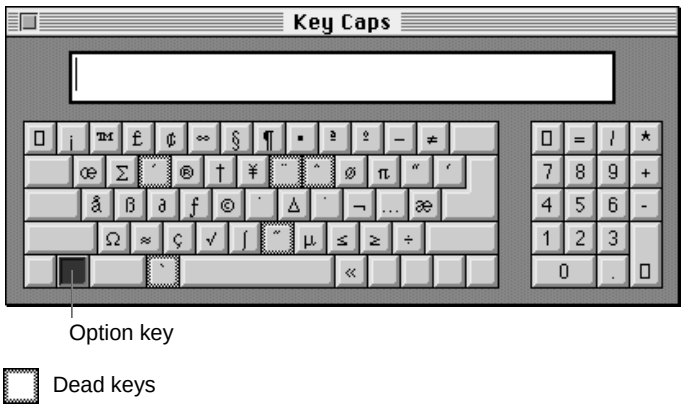
Listing C-4 on page C-31 is a portion of the data from the key-caps resource (`'KCAP'` ID = 1), which is used with the standard ADB keyboard (keyboard type 1, the domestic layout of the Apple Keyboard II). Working with that resource, the Key Caps desk accessory produces the display shown in Figure C-12 when it is used with the standard U.S. keyboard-layout resource (`'KCHR'` ID = 0).

Figure C-12 Key Caps display with key origins



The Key Caps desk accessory provides feedback on using dead keys to produce accented characters. It indicates dead keys with dotted borders, as shown in the Key Caps window in Figure C-13, which shows the U.S. keyboard layout with the Option key pressed.

Figure C-13 Key Caps display of dead keys with Option key pressed



If a dead key is entered, such as the circumflex dead-key combination (Option-I), the display changes to highlight the completer keys for this dead key. The user can press any completer key to generate valid accented character combinations, as shown in Figure C-14. If your application displays keyboards, you should use a similar method of indicating dead keys and completers.

Figure C-14 Key Caps display of completer keys after circumflex dead key has been pressed

Completer keys

Summary of the Keyboard Resources

Assembly-Language Summary

Global Variables

`KbdType`	The keyboard type of the most recently used keyboard
`Key1Trans`	Pointer to key-translation routine (for non-ADB keyboards)
`Key2Trans`	Pointer to key-translation routine (for non-ADB keyboards)