# Technical Note TE23
# International Canceling

**CONTENTS**



This Technical Note describes potential problems canceling operations with the Command-period key sequence and international keyboards.

**[Feb 01 1990]**

---

## Where Did That Key Go?

Canceling an operation, from printing to compiling, has always been done with the key sequence Command-Period. The problem with this is that on some international systems, one needs to hold the Shift key down to produce a period. Many keyboard mappings, including that of the U.S., ignore the Shift key when the Command key is down. In other words, on a system where a period (.) is a shifted character (e.g., Italian) pressing Command-Shift-KeyThatMakesAPeriod does not generate the ASCII code for a period. Instead, the keyboard mapping software generates the ASCII code for the unshifted character. If an application is looking for Command-period to cancel some time intensive operation, and an international user types the shifted key sequence that normally produces a period along with the Command key, the application is going to miss that request unless it takes special precautions.

Back to top

## A Bit Confusing (to me at least)

The solution to this potential international disaster is to strip the Command key out of the modifiers, and then run the key code back through the keyboard mapping software. The trap `_KeyTrans` makes this procedure very easy. `_KeyTrans` takes as parameters a pointer to a `'KCHR'` resource (see M.TB.KeyMapping), a word which contains the keycode and the modifier bits, and a word which serves as a state variable.

One note on the result returned by `_KeyTrans`. *Inside Macintosh* , Volume V-195, The Toolbox Event Manager, states, "ASCII 1 is the ASCII value of the first character generated by the key code parameter." This statement is followed by an illustration (Figure 7 on page V-195) which shows ASCII 1 as the low byte of the high word in the long word result. Although this statement and the accompanying illustration are correct, they have mislead a number of people (me for one).

It is dangerous to expect the character code in one particular word of the long word result. In fact, the architecture of the `_KeyTrans` trap does not specify which word contains the character code in which you might be interested. This is because the `_KeyTrans` trap's primary purpose is to create a package that can be used to build a key-down event, and the Toolbox Event Manager just doesn't care about particular keys. In fact, it is possible to get a result from `_KeyTrans` that contains character codes in both words. This is how dead keys are handled.

But how does one handle a particular character, specifically a period? The strategy adopted in the sample function in this Note is to check both words of the result. If a period exists in either word and the Command key is down, it is counted as a Command-period key sequence.

Now that everything is straight about parameters and results, it's time to look at some sample code. The code fragment which follows ensures that you get that period regardless of the state of the modifier keys.

**MPW Pascal**

```
CONST
  kMaskModifier = $FE00; {need to strip command key from Modifiers}
  kMaskVirtualKey = $0000FF00; {get virtual key from event message}
  kMaskASCII1 = $00FF0000;
  kMaskASCII2= $000000FF; {get key from KeyTrans return}
  kKeyUpMask  = $0080;
  kPeriod = ORD('.');

TYPE
  EventPtr = ^EventRecord;

FUNCTION CmdPeriod(theEvent: EventPtr): Boolean;
VAR
  keyCode     : Integer;
  virtualKey,
  keyInfo,
  lowChar,
  highChar,
  state,
  keyCId      : Longint;
  hKCHR       : Handle;

BEGIN

  CmdPeriod  := FALSE;

  IF ( theEvent^.what = keyDown ) | ( theEvent^.what = autoKey ) THEN BEGIN

    {see if the command key is down.  If it is, get the ASCII }
    IF  BAND(theEvent^.modifiers,cmdKey) <> 0  THEN BEGIN

      virtualKey := BAND(theEvent^.message,kMaskVirtualKey) DIV 256;
    {strip the virtual key by ANDing the modifiers with our mask}
      keyCode := BAND(theEvent^.modifiers,kMaskModifier);
      keyCode := BOR(keyCode,kKeyUpMask);  {let KeyTrans think it was a keyup event,
      this will keep special dead key processing from occurring }
    {Finally OR in the virtualKey}
      keyCode := BOR(keyCode,virtualKey);
      state := 0;

      keyCId := GetScript( GetEnvirons(smKeyScript), smScriptKeys);

      {read the appropriate KCHR resource }
      hKCHR := GetResource('KCHR',keyCId);

      IF hKCHR <> NIL THEN BEGIN
        { we don't need to lock the resource since KeyTrans will not move memory }
        keyInfo := KeyTrans(hKCHR^,keyCode,state);
        ReleaseResource(hKCHR);
      END
      ELSE
        {if we can't get the KCHR for some reason we set keyInfo to the message
         field. This ensures that we still get the Cancel operation on systems where
      '.' isn't shifted.}
        keyInfo := theEvent^.message;

      LowChar := BAND(keyInfo,kMaskASCII2);
      HighChar := BSR(BAND(keyInfo,kMaskASCII1),16);

      IF ( LowChar = kPeriod ) | (HighChar = kPeriod) THEN
           CmdPeriod := TRUE;
    END;
  END;
```

**MPW C**

```
#define kMaskModifiers  0xFE00    // we need the modifiers without the command key
                    // for KeyTrans
#define kMaskVirtualKey 0x0000FF00    // get virtual key from event message for
                    // KeyTrans
#define kUpKeyMask      0x0080
#define kShiftWord      8       // we shift the virtual key to mask it into the
                    // keyCode for KeyTrans
#define kMaskASCII1     0x00FF0000    // get the key out of the ASCII1 byte
#define kMaskASCII2     0x000000FF    // get the key out of the ASCII2 byte
#define kPeriod         0x2E     // ascii for a period

Boolean CmdPeriod( EventRecord *theEvent )
{

  Boolean  fTimeToQuit;
  short    keyCode;
  long     virtualKey, keyInfo, lowChar, highChar, state, keyCId;
  Handle   hKCHR;

  fTimeToQuit = false;

  if (((*theEvent).what == keyDown) || ((*theEvent).what == autoKey)) {

  // see if the command key is down.  If it is, find out the ASCII
  // equivalent for the accompanying key.

  if ((*theEvent).modifiers & cmdKey ) {

    virtualKey = ((*theEvent).message & kMaskVirtualKey) >> kShiftWord;
    // And out the command key and Or in the virtualKey
    keyCode    = ((*theEvent).modifiers & kMaskModifiers)  |  virtualKey;
    state      = 0;

    keyCId     = GetScript( GetEnvirons(smKeyScript), smScriptKeys );
    hKCHR      = GetResource( 'KCHR', keyCId );

    if (hKCHR != nil) {
      /* Don't bother locking since KeyTrans will never move memory */
      keyInfo = KeyTrans(*hKCHR, keyCode, &state);
      ReleaseResource( hKCHR );
    }
    else
     keyInfo = (*theEvent).message;

    lowChar =  keyInfo &  kMaskASCII2;
    highChar = (keyInfo & kMaskASCII1) >> 16;
    if (lowChar == kPeriod || highChar == kPeriod)
      fTimeToQuit = true;

  }  // end the command key is down
}  // end key down event

return( fTimeToQuit );
```

Back to top

## What About That Resource

The astute observer may have noticed that the code example requires that you read a resource. Although this certainly isn't that big of a deal, it is always nice when you can cut down on disk accesses. In System 7.0 a verb is added that can be used to get `_GetEnvirons` to return a pointer to the current `'KCHR'`. The verb is defined and used as follows:

**Pascal**

```
CONST
    smKCHRCache =  38;
```

**C**

```
#define smKCHRCache 38
```

Unfortunately, in system software prior to 7.0, you must use `_GetResource` as demonstrated above to obtain the current `'KCHR'` resource. However, since `_GetEnvirons` always returns zero when passed a verb it does not recognize, you can build System 7.0 compatibility into your application without having to check which system software is running. To do this, you could modify the routines as follows:

**Pascal**

```
CONST {define our own constant until System 7.0 headers ship.  At that point, if you
       have not shipped, you can put in the real constant}
    NewVerb_smKeyCache = 38;
VAR
    KCHRPtr : Ptr;

    KCHRPtr := Ptr(GetEnvirons(NewVerb_smKeyCache ));
    hKCHR   := NIL;  {set to NIL before starting}

    IF KCHRPtr = NIL THEN BEGIN  {we didn't get the ptr from GetEnvirons}
      keyCId := GetScript(GetEnvirons(smKeyScript), smScriptKeys);

      {read the appropriate KCHR resource }
      hKCHR := GetResource('KCHR',keyCId);
      KCHRPtr := hKCHR^;
    END;

    IF KCHRPtr <> NIL THEN BEGIN
      { we don't need to lock the resource since KeyTrans will not move memory }
      keyInfo := KeyTrans(KCHRPtr,keyCode,state);
      IF hKCHR <> NIL THEN
        ReleaseResource(hKCHR);
```

**C**

```
/* again we define our own constant for now */
#define NewVerb_smKeyCache 38

Ptr KCHRPtr;

    hKCHR = nil;  /* set this to nil before starting */
    KCHRPtr = (Ptr)GetEnvirons(NewVerb_smKeyCache );

    IF ( !KCHRPtr ) {
      keyCId = GetScript( GetEnvirons(smKeyScript), smScriptKeys);

      hKCHR   = GetResource('KCHR',keyCId);
      KCHRPtr = *hKCHR;
    };

    IF (KCHRPtr) {
      keyInfo := KeyTrans(KCHRPtr ,keyCode,state);
      if (hKCHR)
        ReleaseResource(hKCHR);
```

Back to top

## References

*Inside Macintosh* , Volume V, The Script Manager

*Inside Macintosh* , Volume V, The Toolbox Event Manager

M.TB.KeyMapping

Back to top

## Downloadables

Acrobat version of this Note (K). Download

---