KEYMASK: A Quick Way to Suppress the Display of Typed Characters

One of the most common questions heard on the FoxForum is “How do I prevent the characters that a user is typing from displaying. I’m writing a logon routine and I don’t want their password to display when they’re typing it.”

In the olden days, about three months ago, you had to write a custom routine that would trap the user’s keypress via INKEY() and manually display another character. This procedure was awkward to implement and, if not carefully trapped for unexpected keystrokes, easily circumvented.

Earlier this summer, however, Ted Roche alerted me to a nifty routine that Scott Mackay has written. KEYMASK is a library that allows you to mask the keys in a GET field from display and display asterisks in their place. In addition to being more convenient (a couple of function calls versus dozens of lines of code), KEYMASK preserves the native user interaction with a GET, including mouse clicks. It’s truly a Cool Tool.

[Insert KEYMASK1.BMP: The KeyMask library allows you to display asterisks in place of characters that the user types, without the nuisances of INKEY() loops.]

KEYMASK functions

KEYMASK consists of two libraries, one each for DOS and Windows. (A Mac version will be forthcoming.) Both libraries contain four functions that provide all the functionality you’ll need.

[Insert KEYMASK2.BMP: KeyMask has DOS and Windows libraries so it can be used in cross-platfrom applications.]

KMClear() clears the keystroke string. This is used when starting the screen or when entering a GET whose display will be hidden.

KeyMask() initiates the masking of keystrokes. In other words, you’d place this function in the WHEN of a GET field. Then, each keystroke typed will be “eaten” and an asterisk will be displayed in it’s place.

KMValue() returns the value of the string that was “eaten” by the KeyMask function. This is useful if you need to do something with the string entered - for example, validating a password.

KMCancel() cancels the masking functionality initiated by KeyMask(). Typically, you’d use this in the VALID of the GET field in order to return the display of typed characters. Otherwise, KeyMask() will continue to display asterisks in place of the actual characters. You’ll also probably want to call KMCancel() in the CLEANUP of the screen in case the user doesn’t execute the GETs valid before exiting the screen’s READ

How to use KEYMASK

Simply issue the command

set library to KEYMASK

when you want to access the four functions listed above. Then, issue KMClear when entering the screen that contains the GET to be masked, issue KeyMask() in the WHEN of the specific GET to be masked, and issue KMClear() in the VALID of the GET to be masked.

You can issue KMValue() at anytime after leaving the GET to capture the value of the string typed (as long as you do so before releasing the KEYMASK library.)

Special Situations

While most developers will typically use KEYMASK just to trap a single GET - in a password situation - you can also trap multiple GETs on a screen, and this situation requires some special handling. An example of this would be a screen where the user is changing their password, and they must enter their existing password, a new password, and then enter their new password a second time.

In this type of situation, the KMClear() function must be called in the WHEN of each GET instead of just in the SETUP of the screen. Also, the KMValue() function would need to be called in each GET's VALID so that one value can be compared to another - when verifying that the new password entered a second time matched the original entry, or if you're enforcing a rule about not letting the old password be used as the new one.

KEYMASK comes with a sample logon program and screen that shows how to use each function - you may be able to plug this sample code right into your application.

Where to find it

KMASK.ZIP is included on this month's Companion Diskette, and can also be found in Library 4 of FoxForum. It's a 32K ZIP file that decompresses into a text file, the two libraries, and the sample LOGON program and screen. It's public domain so there's no registration fee, but do send the author, Scott Mackay, a note telling him thanks for his contribution.