Contents

KeMo Reaction Time Utilities

User's Guide

Overview

The KeMo Reaction Timing Utilities consist of the following parts:

• a timer with 20 microsecond resolution
• polling functions for ADB devices, such as keyboards and mice (+/-1.4 to +/-2.1 msecs accuracy)
• a screen refresh synchronization function for all Macintosh computers
• functions that hide and show the menu bar

These routines are packaged in a library file which can be incorporated into your program. Additionally, a sample program that exercises all of the functions is included with source code in C.

This package was created to facilitate the creation of reaction time experiments on the Macintosh. It improves the accuracy achieved with common Macintosh reaction timing techniques by about a factor of 4.

Requirements

KeMo's ADB (Apple Desktop Bus) polling functions only work on Macs with ADB's. These include all Macs sold by Apple after the Mac Plus. Some incompatibility has been detected with PowerBooks, but other Macs, from Classics to Quadras, all work fine. If KeMoTest does not work with your Mac, I would like to hear from you. If you actually use PowerBooks for experiments, write to me and I will investigate further.

In addition, all functions require System Version 6.0.5 or later.

License

The KeMo set of functions is distributed as “Old World Order/Recession-ware”: if you use them you must send a registration fee of $20 to the U.S. Mail address below <u>unless</u> you fall into one of the following categories:

a) you really cannot afford $20; in this case, send a postcard of your home town

b) you work in a country with non-convertible currency, or you have a hard time getting US dollars; in this case, send one bill of the lowest denomination paper currency of your country

c) you are affiliated with the University of California at San Diego or Harvard University; in this case, send me some e-mail so you will be notified of updates.

I reserve all rights to the KeMo set of functions. You may use them in your own programs, but you may not charge anything for programs that use any KeMo functions. You may not use KeMo functions in any program released as shareware or in other ways such that a user is obligated to pay for using that program. If you are interested in using KeMo in a shareware or commercial application you may contact me at the address below. If you distribute a program that uses KeMo, I expect you will acknowledge the use of KeMo by including the following copyright line wherever you have your own copyright notice:
KeMo Reaction Timing Copyright © 1992 by Dan Costin

This software is offered "as-is," with no warranties. For bug reports, enhancement requests, etc., you may contact me at

U.S. Mail	e-mail
Dan Costin	dcostin@ucsd.edu
P.O. Box 13214	dcostin@ucsd.bitnet
La Jolla, CA 92039-3214	

Accuracy Issues

The KeMo timer has a resolution of about 20 microseconds. The function `KeMoTimerStop()` returns elapsed time rounded off to the nearest millisecond.

ADB (Apple Desktop Bus) keyboards, mice, and other devices, however, cannot report key and mouse button presses nearly as fast. While theoretically (i.e., according to Apple technical specifications) ADB devices can be asked about their keys and buttons every 2 msecs or so, in reality there is some overhead in moving that data around from chip to chip until it can be examined by a program. During normal operation, a Mac checks ADB devices about every 11 msecs or so. If they have some data, that data is then transferred from the ADB device to the Mac. This all leads to a resolution of 16 msecs or more for the detection of events on ADB devices. Furthermore, if a program checks for these events using the EventManager, there is possibly additional time until an event is put on the event queue and until your program detects the event.

KeMo bypasses the higher levels of processing normally used to detect ADB events and achieves a polling rate of slightly less than 4 msecs on most Mac II's (with the appropriate constant correction that results in an accuracy of +/-2 msecs). There is some variation in this rate: on a Mac IIfx the polling rate is less than 3 msecs (accuracy is +/-1.4 msecs), while on a Classic it is around 4.2 msecs (accuracy is +/- 2.1 msecs). On Macs with on-board video in DRAM (e.g. IIsi and IIci without video cards), rather than VRAM, the accuracy can be slightly worse if the program is loaded in the same memory bank as the screen buffer than when it is loaded into a different memory bank. But no matter how fast the computer is, the ADB cannot process requests faster than every 2 msecs or so. That means that if you see a program with a claim for 1 msec accuracy on all Macs, well, it might be too good to be true: with the ADB's 2 msec resolution, a reaction time program can at most claim +/- 1 msec accuracy (which is not the same as 1 msec accuracy: that would be +/- 0.5 msec). Unless, of course, you also get additional hardware, in which case the program does not necessarily use the ADB for responses.

To achieve its higher polling rate KeMo gives up some things, most notably the mouse cursor. When timing keyboard presses (or releases) the mouse is completely turned off, which gives a steady polling rate. When timing mouse button presses, however, the mouse still generates data when it is moved around. This data, depending on how much the mouse is being moved, can decrease the resolution of button detection to over 6 msecs. Unfortunately, even at this slower rate, there is no time to redraw the mouse cursor on the screen, so it stays frozen. For this reason the cursor is always hidden while timing of ADB events is in process (`KeMoSelect()` hides the cursor and `KeMoReset()` shows it).
Important: to get mouse button polling rates that are the same as the keyboard polling rates, do not move the mouse. In an experimental situation where you want to use the mouse button, you might want to take the ball out of the mouse so it cannot be moved. Again, mouse movements do not affect the keyboard polling rate. If you need to use the cursor while checking for mouse button presses, you cannot use `KeMoWait()`: you need to use the Toolbox function

`Button()` (but then you lose the accuracy KeMo gives you).

The Mac has become a complicated computer: it is often hooked up to networks and has many system extensions and INITs all running at the same time as a program, not to mention other programs. Common sense dictates that in order to get the best results when using fast and exact display rates as well as accurate reaction time measurements a program should run with as few other programs as possible. If you have System 7, for example, hold the shift key down during the boot-up process to avoid loading any system extensions, and kill all other applications, like the Finder, at the beginning of your program (a function to do this will be included in future version of KeMo). With System 6, take as many things out of the System file and System folder as you can. This might be a major inconvenience so it might not be a bad idea to have a Mac that is dedicated to running reaction time experiments that require precision in measurement.

Still, even on a Mac that has no other programs running, floppy disk access, SCSI activity (e.g. hard disk access), and network activity during ADB polling introduces some loss of accuracy. These activities only affect accuracy if they occur right when the timer is started or stopped, or when the subject makes a response. Therefore try to avoid writing programs that might allow some of these interfering events to happen at sensitive times. Sometimes disk access can happen even when you don't ask for it (e.g. if you have Virtual Memory); in those cases try to find some other remedy (e.g. turn Virtual Memory off). If your Mac is hooked up to a network, run experiments with the network disconnected. Usually these inaccuracies will be 1 msec or less, but that really depends on the type of extraneous event. For the best advice in you specific situation, ask your local Mac guru what you might have to watch out for.

At the other end of the accuracy spectrum, check out Ulrich, R & Giray, M. (1989). Time resolution of clocks: Effects on reaction time measurement – Good news for bad clocks. British Journal of Mathematical and Statistical Psychology, 42, 1-12. Their main result, as I have seen it reported (I have not seen the paper myself), is that the increased variance due to timing inaccuracies is $\mathbf{m}^2/\mathbf{12}$, where **m** is the resolution of your timer. A clock with 16 msec resolution, then, will increase your standard deviation by

\R(,\F(16^{2},12)) = 4.6 msecs

Please check the paper for yourself, rather than taking my word for it.

Finally, to easily check the accuracy of measurements on your Mac in various environments, run the enclosed KeMoTest program and select the Accuracy option.

Using KeMo in Your Program

To use KeMo in your Think C program, include the KeMo.lib library in your project. You must also include the ANSI or ANSI-small library in order for your project to link properly. For other systems, use KeMo+ANSI.lib. This library file includes the Think C 4.0 ANSI-small library file.

For C programs, include "KeMo.h" in source files that use KeMo functions. For other languages you need to translate the "KeMo.h" file to the appropriate language and include it.

No doubt there will be problems with other languages and other development systems. If so, please contact me at the address on the first page and perhaps we can work out the problems.

A Small Example

Say you would like to implement the following sequence of events in your trial:
  1) a display of the word "dog" for 200 msecs
  2) a display of the word "XXX" for 200 msecs
  3) a display of the word "cat", for which you want to collect reaction time of a key press on the keyboard.

The following pseudo-code might be how you would write your program using KeMo:

```
KeMoParms parms;  /* allocate the parms structure by declaring a local variable */
long err, RT;
char resp;

Macintosh_initialization();
```

```
err = KeMoInit();
Hz = monitor_refresh_rate();  /* find the refresh rate of the current monitor (e.g., by
      calling KeMoSync(1) n number of times and seeing how long it took -- see KeMoTest.c
      for an example */
/* check for error here and for other KeMo functions */

err = KeMoSelect(KeMoKey);  /* RT will be measured from the keyboard */

err = KeMoSync(1);  /* wait for next screen refresh */
display_dog_using_CopyBits();
err = KeMoSynch(Hz/5);  /* wait for 1/5 of a second or as close as possible using the
      current monitor refresh rate */
display_XXX_using_CopyBits();
```

```
err = KeMoSynch(Hz/5);  /* wait another 1/5 of a second */
display_cat_using_CopyBits();
err = KeMoSynch(1);  /* wait until you can be certain the word cat has been displayed
      (but see below) */

err = KeMoTimerStart();  /* start the timer */

err = KeMoWait(KeMoDown, KeMoNoTimeout, &parms);  /* wait for a key press, no timeout,
      and send the address of the parms structure */

RT = KeMoTimerStop(0);  /* get the reaction time */

KeMoReset();  /* turn the mouse back on if there are no more trials */

resp = KeMoCode2Asc(parms.key);  /* translate code to ASCII */

save_responses(resp, RT);
```

Proper methods for screen display of stimuli are beyond the scope of this guide, but you should be aware of the relationship between when you issue a drawing command for an object and when that object actually appears on the screen. It's difficult to measure reaction time from the exact moment when your object has appeared on the screen; for most experiments one can live with a constant measurement error – as long as your stimuli are all at the same screen height, the amount of time it takes the beam to travel from the object to the bottom of the screen (when the refresh occurs) is always the same, so you might as well start measuring your time from the refresh. If you want to be more precise, for an object half-way down the screen add 1/2 the drawing period (1000 msecs/Hz / 2) to the RT as obtained above.

For other examples of how to use KeMo functions, look at the source code for KeMoTest.

KeMoTest

Two versions of the KeMoTest program are provided: one of them, "KeMoTest (w/ debug)" displays information about some of its actions, especially during initialization. This information might be useful if you find that KeMo refuses to work on your Macintosh: if you are having problems, run this program and tell me of any problems it encounters.

The other version, "KeMoTest," is a compiled version of the KeMoTest.c file.

Extending KeMo

If you have an ADB device that you would like to use with KeMo, and you can't figure out how, contact me at the address on the first page and perhaps I will be able to help you. Currently KeMo treats any device with address 2 as a keyboard (this is normally the address of the primary keyboard) and all other devices as mice (i.e., it will only detect up and/or down button transitions for other devices). Run "KeMoTest (w/ debug)" to see what ADB devices KeMo finds on your Mac.

KeMo Functions

Initialization

KeMoInit

Arguments:
- `flags (short int)`: field of bit flags

Returns:
error code (long)

Initializes all functionality in KeMo. This function must be called before using any other functions. `KeMoInit()` must be called after the graphics environment has been initialized, unless the `KeMoNoAlert` flag is set. There is a one to two second delay in this function in order to check the accuracy of the timer and compute the constant correction for `KeMoTimerStop()`. To avoid this delay use the `KeMoNoTCheck` flag (but then you must always call `KeMoTimerStop()` with the `KeMoNoCorrection` flag).

If an error occurs `KeMoInit()` generates two beeps (unless the `KeMoQuiet` flag is set) and attempts to display an alert with an appropriate message (unless the `KeMoNoAlert` flag is set). The alert gives the user two choices: Exit and Continue. If you choose Continue, functions that were not initialized will return error codes, and your system might subsequently need to be restarted, depending on the error.

`Flags` are any combination of the following (to combine use bit-wise OR ('|' in C) or addition):

`KeMoQuiet`	– do not beep on error
`KeMoNoAlert`	– do not display an alert on error
`KeMoAltKeys`	– change the keyboard driver such that it returns different codes for left Shift, Control, and Options keys than for the corresponding keys on the right side of the keyboard; this change is undone when your program exits
`KeMoNoKeys`	– do not try to initialize ADB response functions
`KeMoNoTimer`	– do not try to initialize the timer
`KeMoNoSync`	– do not try to initialize screen synchronization functions
`KeMoNoTCheck`	– do not check the timer and do not compute an ADB constant correction
`KeMoNoMBarInit`	– do not initialize menu bar show/hide functions

If you have no flags to set, use zero.

ADB Device Functions

KeMoSelect

Arguments:
- `device (short int)`: device to select

Returns:
error code (long)

Selects which ADB device input will come from. This function must be called before `KeMoWait()` is called. All ADB devices other than the selected one are turned off. In addition, the mouse pointer is hidden. To turn devices back on and display the mouse pointer, call `KeMoReset()`.

The following devices are defined:

`KeMoKey`	– any keyboard-like device (ADB device address 2)
`KeMoMouse`	– any mouse-like device, including trackballs (ADB device address3)
`KeMoTablet`	– any graphics-tablet-like devices (ADB device address4)
`KeMoDev8` to `KeMoDevF`	– a device with address $8 to $F

Currently only keyboards at address 2 (use `KeMoKey)` and devices that appear to function like mice are supported (this includes trackballs and other mouse-substitute devices). If you have another device that you would like support for, send me some mail at the address on page 1.

KeMoReset

Arguments: none

Returns:
error code (long)

Selects all ADB devices. Must be called to recover operation of any devices turned off with `KeMoSelect()`.

KeMoWait

Arguments:
- `flags (short int)`: what to wait for
- `timeout (long int)`: maximum amount of time to wait in milliseconds
- `parms (KeMoParmsPtr)`: parameter block filled in by KeMoWait that describes the event that caused the end of the wait

Returns:
error code (long)

Waits for either a) an appropriate event (as defined by `flags`) from the device selected with `KeMoSelect()`, or b) the timeout period to expire. Then it fills in the `parms` structure and returns.

Note: `KeMoSelect()` must be called before `KeMoWait()`.

Appropriate events, as passed in the `flags` argument, are one of:

`KeMoDown`	– a down transition (e.g., key press or button press)
`KeMoUp`	– an up transition (e.g., key release or button release)
`KeMoUpDown`	– either an up or a down transition

The timeout's maximum value is approximately 35 minutes. If no timeout is desired, use the value `KeMoNoTimeOut`.

The `parms` structure contains the following fields:

`key (byte)`	– the keyboard code of the key involved
`key2 (byte)`	– if two keys are accessed simultaneously, the keyboard code of the second key
`updown (byte)`	– one of `KeMoUp` or `KeMoDown`, describing the direction of the event
`updown2 (byte)`	– the direction for `key2`, if active

When the function returns because the timeout period expired, the `updown` field contains `KeMoTimedOut`. If there is no second simultaneous key, the `key2` field contains $FF. To translate keyboard codes to the corresponding ASCII characters, see the `KeMoCode2Asc()` function below.

You must allocate a `parms` structure and send its address (a pointer to it). KeMo does not allocate this structure for you.

KeMoCode2Asc

Arguments:
- `code (byte)`: the keyboard code

Returns:
ASCII code representation (byte)

Given a keyboard code returned by `KeMoWait()`, it returns an ASCII code. Keyboard characters are translated into their unshifted representation. Keypad numbers 0 to 9 are converted to ASCII codes 0 to 9 (defined in `KeMo.h` as K0 to K9), and function keys F1 to F15 are converted to ASCII codes 11 to 25 (defined in `KeMo.h` and F1 to F15). Upper case letters represent special keys, as follows:

ASCII	Key	C #define	ASCII	Key	C #define
A	Keypad-/	`KSLASH`	S	Shift (left)	`SHIFTL`
B	Tab	`TAB`	T	Control (left)	`CTLL`

C	Command	`COMMAND`	U	Page Up	`PGUP`
D	Delete	`DELETE`	V	Down Arrow	`DOWN`
E	Esc	`ESC`	W	Page Down	`PGDOWN`
G	Power	`POWER`	X	Del	`DEL`
H	Help	`HELP`	Y	Home	`HOME`
K	Keypad-Enter	`KENTER`	Z	End	`END`
L	Caps Lock	`CAPS`	^	Up arrow	`UP`
M	Keypad-'-'	`KMINUS`	<	Left arrow	`LEFT`
N	Clear	`NUMCLEAR`	>	Right arrow	`RIGHT`
O	Option (left)	`OPTIONL`	$	Option (right)	`OPTIONR`
P	Keypad-.	`KPERIOD`	@	Shift (right)	`SHIFTR`
Q	Keypad-=	`KEQUAL`	#	Control (right)	`CTLR`
R	Return	`RETURN`			

Keys not on the Apple Extended Keyboard are converted to '?'. Keyboard codes out of range are converted to '!' (this will happen if you try to convert the keyboard code from `KeMoWait()` when it times out).

The Option, Shift, and Control keys can be differentiated left from right by using the `KeMoAltKeys` flag with `KeMoInit()`. If not differentiated, the left codes are generated for both left and right keys.

Do not use this function with the return values from the Toolbox `GetKeys()` call. The two representation of the keyboard are different.

Timer Functions

KeMoTimerStart

Arguments: none

Returns:
error code (long)

Starts the millisecond timer. Will return an error code (`KeMoErrorTimerRunning`) if the timer is already running, but it will still restart the timer.

KeMoTimerStop

Arguments:
- `flags (short int)`: processing options

Returns:
time value in milliseconds (long)

Stops the millisecond timer and returns the number of milliseconds since the last time `KeMoTimerStart()` was called. If the return value is negative, it is an error code.

If the `KeMoNoCorrection` flag is not present, a constant correction is subtracted from the actual time before that value is returned. This correction is computed in `KeMoInit()` (unless the `KeMoNoTCheck` flag is set there) and consists of the constant time it takes the ADB device to transfer its information once something has been activated (about 1.6 msecs) plus one half of the time it takes to poll an ADB device (therefore the accuracy is expressed as +/- one-half the polling rate). If you are not timing an ADB device response, you must use the `KeMoNoCorrection` flag for the appropriate result.

`Flags` may be set to:

`KeMoNoCorrection` - do not subtract the constant correction normally used for timing ADB events

If you have no flags to set, use zero.

KeMoDelay

Arguments:
- `time (long int)`: amount of time to delay processing by in milliseconds

Returns:
error code (long)

Waits the specified number of milliseconds and returns. This function may be called at any time, even if the timer is running, without affecting other timing operations.

Synchronization Functions

KeMoSync

Arguments:
- `when (short int)`: how many screen refreshes to wait before returning (1 means return on the first refresh that comes along)

Returns:
error code (long)

Waits until the main screen (the screen with the menu bar on it) is about to be redrawn and returns. Useful for drawing to the screen without flicker, as well as knowing exactly when something drawn to the screen has shown up.

Menu Bar Functions

KeMoHideMBar

Arguments: none

Returns:
error code (long)

Removes the menu bar from the screen.

KeMoShowMBar

Arguments: none

Returns:
error code (long)

Restores the menu bar.

Testing Functions

KeMoAccuracy

Arguments: none

Returns:
resolution in microseconds (long)

Computes the rate at which a selected ADB device is polled. Once this function is called, wait a few seconds and press the appropriate device (e.g., keyboard or mouse). Note that the returned value is in microseconds, not milliseconds.

You must call `KeMoSelect()` at some point before you call this function.

KeMoTimerTest

Arguments: none

Returns:
resolution in microseconds (long)

Computes how steady the timer is while polling is active by comparing it with the system clock over a period of ten seconds. There is an initial delay of 1/2 second at the start, after which keyboard polling is initiated. A key must be pressed in the ensuing 10 seconds. If no keys are pressed, there is a beep 7 seconds after the start and another at 10 seconds. Pressing a key after 10 seconds will result in an error code being returned (a negative number).

Note that some older VIA-based timers would not function correctly in this type of test. To avoid inaccurate timing, do not use other timers with KeMo's ADB polling routines unless you have tested them and found that they are not affected by KeMo functions.

Error Codes

Error codes, as listed in KeMo.h, have the following meanings:

-501 KeMoErrorVeryOldSystem	the current System Version is much too old for KeMo functions (older than 4.1)
`-502 KeMoErrorOldSystem`	the current System Version is too old; KeMo needs System Version 6.0.5 or later
`-503 KeMoErrorNotADB`	KeMo cannot find any ADB devices
-504 KeMoErrorNotInitialized	a KeMo function other than `KeMoInit()` was called before `KeMoInit()`

-505 KeMoErrorBadArgs	check your arguments in this function call
`-506 KeMoErrorUnsuccessful`	the requested function could not be completed
`-507 KeMoErrorNoSuchDevice`	the device you requested to be selected does not exist
`-508 KeMoErrorTimerRunning`	the timer is already running – this is only a warning
`-509 KeMoErrorTimerNotRunning`	cannot stop the timer if it is not running
`-510 KeMoErrorTimerNotInit`	cannot call timer routines without first calling `KeMoInit()` (check its return code as well – initialization could have failed)

`-511 KeMoErrorTimerOff`	initialization failed since it detected that the timer is not accurate on this machine
`-512 KeMoErrorImproperTest`	an accuracy test was not performed properly
`-513 KeMoErrorNoHeapAlloc`	needed heap area allocations could not be made; reduce the number of system extensions (or INITs) and reboot
`-514 KeMoErrorNoSync`	cannot figure out how to synchronize to the screen on this machine
`-515 KeMoErrorGestalt`	error calling the Toolbox `Gestalt()` function: most likely caused by an unknown Mac type
`-516 KeMoErrorNoCorrection`	`KeMoTimerStop()` cannot return a corrected value since a correction was not computed by `KeMoInit()`; call `KeMoTimerStop()` with the `KeMoNoCorrection` flag set
`-517 KeMoErrorUnknownDevType`	the device type of the device to be selected is not currently supported
`-518 KeMoErrorOneDevOnly`	currently only one device at a time can be selected
`-519 KeMoErrorMBarHidden`	a call to `KeMoHideMBar()` was made while the menu bar was already hidden
`-520 KeMoErrorMBarShows`	a call to `KeMoShowMBar()` was made while the menu bar was showing