InputLogger: General purpose logging of keyboard and mouse events on an Apple Macintosh

Shari Trewin

University of Edinburgh

Department of Artificial Intelligence

80 South Bridge

Edinburgh

EH1 1HN

Scotland

Phone: 44 131 650 2725

Email: *shari@aisb.ed.ac.uk*

Suggested running head: Macintosh Input Logging

ABSTRACT

Event logging, particularly logging of event timing information, is often used in human-computer interaction research in investigations of the ways in which people use computers, and in the evaluation of input devices and applications. This paper describes *InputLogger*: a low-level input event recorder for the Apple Macintosh†. It differs from other keystroke loggers in that it records accurate timing information for all keyboard and mouse events, while being application independent. It is capable of logging any Macintosh session. *InputLogger* has been used to gather data on the difficulties experienced by people with motor disabilities using ordinary keyboards and mice. It would be appropriate for many other experimental applications.

Input Logging

Timed input event logging is appropriate wherever computers are used to investigate human behaviour involving time, such as reacting to stimuli, making precise movements, or interacting with a computer application or input device. Timing and/or log information is used in many research areas, including evaluating the usability of different pointing devices (Kabbash *et al*., 1993; MacKenzie *et al*., 1991; Riviere & Thakor, 1996), the evaluation of keyboard designs (Smutz *et al*., 1993), investigating theories of human movement control (Gillan *et al.,* 1990; Walker *et al*., 1993), looking at individual differences in typing characteristics (Leggett *et al*., 1991), and evaluation of application programs and their interfaces (Brewster *et al*., 1994; Roberts & Moran, 1983).

Although the majority of studies are limited to either the mouse or the keyboard, in some areas, such as application interface evaluation, or the evaluation of a whole computer setup, it is necessary to look at the interaction between mouse and keyboard usage. Examples include the work of Douglas and others, who looked at the significance of the time needed to move between the mouse and the keyboard (Douglas & Mithal, 1994), and the author's own investigations into the usability of standard input devices for people with motor disabilities (Trewin, 1996).

While some experimenters have used manual timing and logging mechanisms (Roberts & Moran, 1983), this has generally been due to the lack of availability of general purpose logging mechanisms. Roberts and Moran noted that they used stopwatch timings because: ``not everyone has access to an instrumented editor or videotape setup''. Where available, an automatic recording is superior in accuracy and detail to a human recording, particularly where timing information is concerned. Automatic logging also frees the experimenter to make observations of events of particular interest, such as errors made by subjects.

Unfortunately, the majority of existing packages for recording Macintosh input events are limited to either the keyboard or the mouse, and many are built in to a specific application. This is because they are often custom-written for a specific purpose, such as recording the process of editing a document, or the time taken to point to a given target.

Because of the time and effort required to develop logging software, custom-built loggers tend to record the minimum amount of information necessary for the specific experiment for which they are developed. As a result, they are not as reusable as might be hoped.

The *InputLogger* program attempts to provide a reusable, general purpose logging mechanism for the Macintosh, by recording detailed information about both mouse and keyboard input events. This detail includes key down and key up times for all keys, including control keys, mouse down and mouse up times, and a description of the path taken by the mouse during pointing movements. These logs can then be tailored to suit specific experiments by providing an appropriate filter.

As an introduction, the following section describes the aspects of the Macintosh architecture relevant to recording events. The paper goes on to give a review of existing Macintosh logging software, followed by an overview of *InputLogger*. The benefits and costs of application independence are discussed. The following section describes the implementation of *InputLogger*, covering some of the difficulties in providing accurate logging information on a Macintosh, and how they have been tackled. Those interested in trying the software are referred to the final section, which gives details of its availability.

Macintosh Input Events

Before describing Macintosh logging software, it is worth reviewing the Macintosh architecture, and the way input events are represented internally.

When using a keyboard or mouse, there are a limited number of inputs that a user can make. The Macintosh operating system recognises and processes the following set of input events:

- `KeyDown`: a non-modifier key has been pressed down.
- `KeyUp`: a non-modifier key has been raised.
- `KeyRepeat`: a non-modifier key is being held down, generate

another instance of the appropriate character.

- `MouseDown`: the mouse button has been pressed down.
- `MouseUp`: the mouse button has been raised.

`KeyUp` events are often suppressed or ignored, as the majority of applications do not require them.

A time is associated with each event, using the system clock measure of the number of ticks since system startup. A tick is a sixtieth of a second. In addition, a key code is associated with keyboard events.

Mouse movements do not generate events, but the current mouse position is recorded with every internal event, including `NULL` events. This allows the mouse path to be tracked by checking the mouse position for every internal event.

In addition, pressing of modifier keys such as *Shift* or *Command* does not cause any keyboard events. Instead, internal flags are set. These modify the key code reported for other key presses, so that each alphanumeric key is modified in different ways by the flag settings in place when it was pressed down. This treatment of modifier key presses makes it difficult to record accurate information about them. The approach taken in *InputLogger* is described later in this paper.

The *InputLogger* program aims to record as accurate a picture of input device usage as possible. This means logging all of these input events, plus mouse paths and presses of modifier keys.

Existing Macintosh Logging Software

Several tools for recording keystrokes and/or mouse movements for the Macintosh already exist.

For example, JEdit (Eklundh & Kollberg, 1996) is a dedicated text editing application, which records those input events relevant to the text editing process: key down and mouse up. Key repeat events are recorded as key down events. It produces log files with timings of keystrokes to within 0.1 sec. Positions of mouse clicks and selections are recorded as positions within the text, rather than screen positions which would be more difficult to interpret. A mechanism for playing back the recordings made by JEdit is also available.

Another logging text editor is TeachText+, developed by Steve Lang of Loughborough University of Technology (Lang, 1995). It is an extension of the TeachText editor which records timings of key down and mouse down events, to the nearest tick. Mouse clicks, double clicks and drags are recorded simply as 'mouse events', with a start and end position related to the file being edited.

ProtoTymer (Miller & Stone, 1990) is a system to collect keystrokes and mouse events from a Hypercard application. It allows playback of these events, but imposes severe restrictions on the amount of data that can be stored.

An alternative approach to logging is to make a recording of the screen, which can be played back. This is exemplified by MediaTracks, the Farallon screen recorder. This approach can produce very large and unwieldy log files. In MediaTracks, timing information can only be included by putting a clock on the screen, so accurate split-second timing is not possible.

More general event trappers do exist. Now Utilities provides facilities for recording the use of applications, but is limited to keystrokes only, and provides no timing information. MacroMaker is part of the standard Apple distribution, and will record short sequences of keystrokes and mouse clicks to be replayed on demand. It is, however, only suitable for short event sequences, as it has a noticeable effect on application response times. Timing information can be included but causes very large log files to be produced.

The majority of these loggers are concerned with recording events that affect the system, rather than the user's original actions. Most do not distinguish between key presses and key repeats, or keep a record of mouse movements. Key up events are generally not significant, and some systems omit mouse down or mouse up events, even when the mouse is being dragged. Many systems provide a video or other mechanism for playing back events in the order they occurred, but timing information is often crude, and again the mouse path is generally omitted. The author is not aware of any system which provides full information about the mouse and keyboard usage. The *InputLogger* program is intended to fill this gap.

InputLogger

InputLogger is a system extension/control panel combination which, once installed, generates ASCII log files recording the input presented over a given period. Logging is switched on and off via the control panel. When actively logging events, *InputLogger* records events regardless of what application is in use, and users may switch between applications as often as desired.

Because *InputLogger* is implemented at a low level in the system software, it is fast and unobtrusive. The use of separate processes for reading and writing events allows event records to be stored up during busy periods, such as when the user is typing fast, and written to the log file during a pause. This helps to minimise the effect of logging on the response time of the running application.

The events recorded in the log file are exactly those recognised by the Macintosh operating system: `KeyDown`, `KeyUp`, `KeyRepeat`, `MouseDown`, and `MouseUp`, with the addition of one new event type: `MouseMove`. A `MouseMove` event is recorded whenever the mouse changes direction by more than 10 degrees. This provides an accurate yet compact record of the mouse path taken between mouse clicks, including small-scale shake and large scale positioning movements. The default value of 10 degrees can be altered on the *InputLogger* control panel. For reasons explained in the description of the logging mechanism itself, `MouseMove` events are not recorded while the mouse button is held down. Presses of modifier keys are recorded as `KeyDown/KeyUp` pairs, in the same way as other key presses. `KeyRepeat` events are not generated for modifier keys.

The default behaviour of *InputLogger* is to record `KeyUp` events. An alternative version suppresses these events.

An example log file is shown in Figure 1. Each log file entry is of the form:

```
<nLost> <time> <eventType> <eventData>
```

- `<nLost>` – is a check for internal logging failures, and should remain zero (see the description of the logging mechanism for more details).
- `<time>` – the time at which the event occurred. Time is measured as the number of *ticks* since system startup. A tick is a sixtieth of a second.
- `<eventType>` – specifies what event has occurred. One of: `KeyDown (kd)`, `KeyUp (ku)`, `KeyRepeat (kh)`, `MouseDown (md)`, `MouseUp (mu)`, `MouseMove (mm)`.
- `<eventData>` – For keyboard events, the event data recorded is the hexadecimal ASCII code for the relevant key. Unused ASCII codes are employed to specify keys such

as *Shift* which have no ASCII code of their own. For mouse events, event data is the mouse position on the screen, recorded as separate x and y co-ordinates.

InputLogger was developed in Ansi C using the Metrowerks' CodeWarrior[†] development system (Metrowerks Inc., 1993) on a PowerMacintosh 6100/66. The default configuration has been used on both this machine and a Macintosh 475 8/160 with no noticeable performance degradation. On the PowerMacintosh 6100/66, *InputLogger* has proved adequate for recording five minutes of continuous random character typing, including `KeyUp` events, at a rate of 150 words per minute. It should therefore be adequate for even the fastest touch typist.

The size of the log files generated in a given session depends upon the amount of input generated during the session, rather than the length of the session. One minute of continuous mouse movement will generate a log file of approximately 20k, while 1000 typed characters will be recorded in a 46k log file, assuming `KeyUp` events are recorded. Only 23k would be required if the option to ignore KeyUp events were used. Should the logging mechanism be used for several hours of keyboard or mouse activity, the upper limit on the size of the log file produced is dependent on the memory available: *InputLogger* imposes no limit.

Application Independence

The use of an application-independent logging mechanism has several advantages over loggers that are built into a custom-designed environment. The ability to log the use of any word processor, graphics package or other Macintosh application provides the experimenter with freedom to study any existing application without the need to modify it, or to design tests using any package or combination of packages.

Using the same mechanism to log different types of task facilitates reliable between-task comparisons. It also allows analysis of use of the operating system itself. Actual computer usage in a real work environment can be recorded directly without interrupting the user or requiring them to run special versions of any software.

The major disadvantage of general purpose logging mechanisms is a technical limitation imposed by the Macintosh operating system. This makes it impossible to record the path taken by the mouse while dragging. For mouse drag path recording, an application-specific logging mechanism would be necessary. *InputLogger* simply records the times and positions of the mouse down and mouse up events of a drag.

It is sometimes also useful to incorporate application-specific information into a log, so that input events can be viewed at a higher level, such as a click on a specific button. It would be possible to extend *InputLogger* to incorporate facilities for higher level event recording, but this would slow the logger. A more efficient solution is to use post-processing of the log files to produce a higher-level trace.

The Logging Mechanism

InputLogger consists of a control panel and a system extension. The extension is a patch on the `SystemEvent` internal Macintosh routine. It traps keyboard and mouse events of interest and records them. The control panel provides an on-off mechanism, writes events into the log file, and also directly records events of interest that the system extension could not catch.

The control panel and system extension communicate using Apple's Audit† library. This provides an unobtrusive event tracing facility, usable from all types of Macintosh code segments. Events can be written into a store and read from the store. In *InputLogger*, events are written by the system extension, and read by the control panel. *InputLogger* can set the size of Audit's internal store and the behaviour of the library when the store becomes full. If the internal store is full, *InputLogger* overwrites the oldest event in the store. Audit warns whenever events have been lost, and *InputLogger* writes the number of events lost into the log file. If events are being lost, non-zero values will appear in the first column of the log file (see Figure 1). The Audit library does not allow dynamic adjustment of the size of the internal store. If events are lost, the solution is to restart the program with a larger internal store by changing a constant in the header file and recompiling. While inconvenient, this is unlikely to be necessary, since the default used by *InputLogger* (64 entries) has proved adequate for recording typing rates of 150 words per minute.

Logging alphanumeric keystrokes and mouse events

When logging is switched on, the control panel opens a new log file and enables the Audit record. While the Audit record is enabled, the system extension examines all events reported to the `SystemEvent` routine. Whenever an alphanumeric key is pressed, a `KeyDown` event is generated. The system extension puts the event, including the time at which it happened, and the key that was pressed, into the Audit record. The control panel reads the event from Audit, along with a count of how many events, if any, have been lost, and writes this information into the log file. When the key is released, a `KeyUp` event occurs and is treated in the same way. `MouseDown` and `MouseUp` events are recorded along with a time, and the mouse position for the event.

Logging mouse movements

Mouse movements do not cause input events, but the current mouse position is recorded along with every Macintosh event that is reported to `SystemEvent`. This allows the system extension to keep track of mouse movement by comparing the mouse position of the current and previous event. Writing out a `MouseMove` event every time the mouse position changed would produce extremely large log files, and so *InputLogger* explicitly records a mouse movement only when the direction in which the mouse is moving changes by at least the number of degrees specified on the control panel. Then, the mouse position recorded *before* the change of angle is written to the log file, and the angle of movement between that position and the current mouse position is stored as the new direction of movement.

This, however, is not enough, as the system extension can only look at the mouse position whenever some interface event occurs. An accurate record of mouse movement requires the mouse position to be checked while no interface events are occurring. When no events are occurring, the Macintosh generates `NULL` events. These have the same format as normal events, including a record of the mouse position, but are treated differently. Most importantly, they are not reported to the `SystemEvent` function, and so are not trapped by *InputLogger*'s system extension. Instead, the control panel catches these events and tracks the mouse position in the same way as the system extension does, inserting `MouseMove` events whenever necessary.

Logging modifier key presses

As mentioned previously, modifier keys such as *Shift* or *Command* must be treated differently to alphanumeric keys, as no event is generated when they are pressed. A similar mechanism to that used for mouse movements is employed - every time an event is reported to the system extension, the status of the modifier keys is checked. Every time a change has occurred the appropriate `KeyDown` or `KeyUp` event is lodged with Audit. As with mouse movements, modifier keys must be checked on `NULL` events by the control panel itself. This method provides as accurate a picture of modifier key presses as possible, although it is still less accurate than recording of alphanumeric key presses.

There is one key that cannot be recorded in the same way as the others: the *Caps Lock* key. This key is similar to other modifier keys in that no events are generated when it is pressed. Instead, its status is recorded in internal structures and can be read. However, because it is a locking key, when it is pressed, the appropriate status bit is set, and this bit remains set until the key is pressed again. The length of time for which the key was actually pressed down each time cannot be established by the logging software. Instead, the log shows a single `KeyDown` or `KeyUp` event for each press of the key.

Log file idiosyncrasies and limitations

Events written into the log are generated from two different sources: the control panel and the system extension. These both keep a separate record of the mouse movements and control key positions. As a result, sometimes mouse events are written to the file out of order, and control key movements are occasionally duplicated. Both the control panel and the system extension use the same event timestamp mechanism, and so a simple sort of the log files is sufficient to provide a correctly ordered set of events. Duplicated events can be ignored.

With regard to recording mouse drag movements, *InputLogger* is limited by the behaviour of the Macintosh environment. When the mouse button is pressed down, no events are processed until the mouse is released -- if a key is pressed during this period it will not appear on the screen until the mouse is released.[1] Even `NULL` events are not reported. This means that *InputLogger* cannot gain control of the machine in order to check the status of control keys and the mouse position until the mouse button is released. Consequently, the path taken by the mouse while dragging, and changes in control key status while dragging, cannot be determined.

Potential Extension Conflicts

The use of Macintosh system extensions, while providing a very efficient logging mechanism, also introduces the potential for conflicts between *InputLogger* and other system extensions. Any extension using the Gestalt selector 'klog' will clash with *InputLogger*. Similarly, any program writing to a file with the same path name as the log file may also clash. No other specific clashes are known, tail patching is not used, and other instances of the Audit library will not cause problems. A Gestalt selector or file name conflict can be avoided by changing the Gestalt selector or file name used by *InputLogger,* recompiling and reinstalling, or by disabling the extension which clashes.

Availability

Interested readers can obtain a copy of *InputLogger* from the author, who would welcome feedback and comments. The software, including source code, can be downloaded via ftp from

[1]The highlighting of selected areas during a mouse drag is performed by the Macintosh operating system function: `TEClick` (Apple Computer Inc., 1985). `TEClick` returns only when the mouse button is released, so patching this function would not help. There is a mechanism by which applications can register a function to be called periodically by `TEClick`, but there is no way for *InputLogger* to access this facility for a given application.

ftp.dai.ed.ac.uk/pub/user/shari/InputLogger.sit.hqx, or accessed via the author's web page at http://www.dai.ed.ac.uk/students/shari, and can also be made available on disk. For the disk version, a fee of £5 (US $10) is requested to cover postage, package and time.

† Metrowerks is a registered trademark of Metrowerks Inc. CodeWarrior is a trademark of Metrowerks Inc. The Audit library is copyright 1992-3 Apple Computer Inc. Apple is a registered trademark of Apple Computer Inc. Macintosh is a trademark of Apple Computer Inc.

References

APPLE COMPUTER INC. (1985). *Inside Macintosh*, **1**, Addison Wesley.

BREWSTER, S., WRIGHT, P., & EDWARDS, A. (1994). The design and evaluation of an auditory-enhanced scrollbar. *Proceedings of CHI*, 173-179, New York, ACM.

DOUGLAS, S., & MITHAL, K. (1994). The effect of reducing homing time on the speed of a finger-controlled isometric pointing device. *Proceedings of CHI*, 411-416, New York, ACM.

GILLAN, D., HOLDEN, K., ADAM, S., RUDISILL, M., & MAGEE, L. (1990). How does Fitt's law fit pointing and dragging? *Proceedings of CHI*, 227-234, New Yord, ACM.

KABBASH, P., MACKENZIE, I., & BUXTON, W. (1993). Human performance using computer devices in the preferred and non-preferred hands. *Proceedings of INTERCHI*, 474-481, New York, ACM.

LANG, S. (1995). *TeachText Plus* [computer program], personal communication.

LEGGET, J., WILLIAMS, G., USNICK, M., & LONGNECKER, M. (1991). Dynamic identity verification via keystroke characteristics. *International Journal of Man-Machine Studies*, **35**, 859-870.

MACKENZIE, S., SELLEN, A., & BUXTON, W. (1991). A comparison of input devices in elemental pointing and dragging tasks. *Proceedings of CHI*, 161-166, New York, ACM.

METROWERKS INC. (1993). *CodeWarrior 6*. [computer program] The Trimex Building, Route 11, Mooers, NY 12958, USA.

MILLER, D., & STONE, A. (1990) ProtoTymer: human performance instrumentation for Hypercard prototyping. *Proceedings of CHI*, 466.

RIVIERE, C., & THAKOR, N. (1996). Effects of age and disability on tracking tasks with a computer mouse: Accuracy and linearity. *Journal of Rehabilitation Research and Development*, **33(1)**, 6-15.

ROBERTS, T., & MORAN, T. (1983). The evaluation of text editors: methodology and empirical results. *Communications of the ACM*, **26**, 265-283.

EKLUNDH, K., & KOLLBERG, P. (1996). Computer tools for tracing the writing process: from keystroke records to s-notation. In RIJLAARSDAM, G., COUZIJN, M., & VAN DEN BERGH, H. (eds), *Theories, Models and Methodology in Writing Research*. Amsterdam University Press, Amsterdam.

SMUTZ, P., SERINA, E., & REMPEL, D. (1994). A system for evaluating the effect of keyboard design on force, posture, comfort and productivity. *Ergonomics,* **37(10)**, 1649-1660.

TREWIN, S. (1996) A study of input device manipulation difficulties. *Proceedings of the Second Annual ACM Conference on Assistive Technologies*, 15-22, USA, ACM.

WALKER, N., MEYER, D., & SMELCER, J. (1993). Spatial and temporal characteristics of rapid cursor-positioning movements with electromechanical mice in human-computer interaction. *Human Factors*, **35**, 431-458

Figure 1: *Inputlogger:* Example Log File

```
                  mouse co-ords
            event type
         time
events lost

  0  87868 mm  306 182
  0  87879 mm  302 181
  0  87879 md  302 181
  0  87898 mu  302 181
  0  88611 kd  F2
  0  88728 kd  54
  0  88738 ku  54        key
  0  88807 ku  F2      pressed
  0  88898 kd  68
  0  88905 ku  68
  0  88957 kd  65
  0  88963 ku  65
  0  88983 kd  72
  0  88990 ku  72
  0  89011 kd  65
  0  89011 kd  64
  0  89019 ku  65
  0  89021 ku  64
  0  89236 kd  20
  0  89244 ku  20
```