Behavior Research Methods, Instruments, & Computers
1999, 31 (3), 410-415

A tutorial on creating logfiles for event-driven applications

GREG BREINHOLT and HELMUT KRUEGER
Swiss Federal Institute of Technology, Zurich, Switzerland

This paper describes the practical steps necessary to write logfiles for recording user actions in event-driven applications. Data logging has long been used as a reliable method to record all user actions, whether assessing new software or running a behavioral experiment. With the widespread introduction of event-driven software, the logfile must enable accurate recording of all the user's actions, whether with the keyboard or another input device. Logging is only an effective tool when it can accurately and consistently record all actions in a format that aids the extraction of useful information from the mass of data collected. Logfiles are often presented as one of many methods that could be used, and here a technique is proposed for the construction of logfiles for the quantitative assessment of software from the user's point of view.

LOGFILES

Logfiles historically evolved from the testing measures that were necessary with the early programming languages (Hetzel, 1985). They were introduced, for technical reasons, to record the sequence of functions that were called within a program, to assess its efficiency and to help in the debugging. They gave the programmers a detailed record of the flow through a program. The important factors often were time and memory used, and the values for these were sampled and stored. Testing was routinely associated with engineering and manufacturing processes, and it was quite natural to see it take shape as part of the software development process.

Out of logging for purely technical evaluation arose the need to log for knowledge of what the user was doing. One of the earliest user interface guidelines states, "User records will permit assessment of performance and improvement of user interface design" (Smith & Mosier, 1986, p. 333). Unfortunately, exactly how to implement an effective logging method and how to use the resulting data to improve user interface design is not documented. This shift from a technocentric approach to a user-centered approach has also shifted the purpose of the logfile from being a record of where the program has been to being a record of where the user has been and what he or she has done there. This change has not always been implemented in the most efficient manner (from the authors' personal experience), and occasionally a logfile is created that cannot actually be used to analyze the user actions, but only details the interior workings of the program. Data logging is often implemented in experiment controller hardware and software to enable the subsequent analysis (Palya & Walter, 1993).

Developing from basic logging came the capture and playback method (also called capture/replay). It is one of the most popular commercial tools for interface evaluation (Beizer, 1990). During capture, all the interactions between the user and the software are recorded in a script file. This includes all messages displayed on the screen, all keypresses, and all movements and actions with the mouse. Later, this script file can be played back to the original software, to simulate the presence of the user. The programming needed to modify software so that it generates such scripts is often a difficult and time-consuming task, and recently, noninvasive methods that do not require that the original software be modified have been developed (Nesi & Serra, 1995). Although the capture and playback method provides a good visual record of the user's actions, it is often not suited for statistical analysis, because the script file is structured to aid the replay and not the analysis of the data.

Logfiles can be analyzed by many statistical packages, providing quantitative data about times to complete actions, frequencies of actions, and so forth. There are more complex methods, such as *maximal repeating pattern* analysis (Siochi & Ehrich, 1991; Siochi & Hix, 1991). This technique uses an algorithm to detect repeated user actions in data files. The hypothesis is that repeated sequences of user actions are of interest to the evaluator and may indicate problems with the user interface.

When compared with other methods, such as verbal protocol analysis, questionnaires, or interviews, logfiles have several advantages and, unfortunately, several disadvantages (Henderson, Smith, Podd, & Varela-Alvarez, 1995; Jeffries, Miller, Wharton, & Uyeda, 1991). The advantages stem mainly from the ease with which some sort of quantitative results can be obtained. Although large

Correspondence concerning this article should be addressed to G. Breinholt, Emmasingel 24, Building HWD, Box 218, 5600 MD Eindhoven, the Netherlands (e-mail: greg.breinholt@philips.com).

amounts of data may be produced, the data evaluation is often quicker than other methods (Yamagishi & Azuma, 1987); there may, however, be a need for a specialized analysis tool (Hoiem & Sullivan, 1994). The limitations of logging come from the nature of the data, which are purely objective and contain no subjective user preferences (Henderson et al., 1995; Hoiem & Sullivan, 1994; Jeffries et al., 1991; Yamagishi & Azuma, 1987). Logfiles must record the user's actions in terms of what, where, and when, to later answer the why?

EVENT-DRIVEN APPLICATIONS

The general trend to move from text-based command interfaces to interaction through direct manipulation and graphical interfaces has radically changed the dialogue that occurs between user and computer. The locus of control is finally moving in favor of the user, whereas the computer acts after, and waits for, the user response. This gives the user more freedom when interacting; at any one time, there are increasing numbers of objects that respond when the user clicks the mouse, fields that change when text is entered via the keyboard, commands that may be sent through character-based shortcuts, and in the future, voice-based recognition of spoken commands. The program must accept these interactions and then respond appropriately, while keeping a record of these inputs. The choice of user actions, termed *events*, should drive the program, keeping the user in control of the flow through the task at hand.

Although there are many possible user events, they must all be recorded in such a manner that there is a consistency and reliability to their format that enables the analysis. It is this structured format that greatly enhances the value of the logfile when sifting through the often large quantity of recorded data.

IMPLEMENTED EVENTS

Two of the most popular development tools for prototyping and writing experiment applications are Visual Basic (VB) running under Windows 95/NT and HyperCard on the MacOS. VB uses a version of the standard BASIC language, whereas HyperCard uses its own HyperTalk, a language with a simplified English syntax. These development tools are similar in their choice of programming paradigm: They are object based and event driven. The languages use graphical objects (such as buttons, menus, text fields, etc.) that respond to certain user events. These objects or controls are named OLE Custom Extensions (OCX) or ActiveX Controls in VB. They may respond to many user events, although normally only a subset is implemented for each control. In VB, the possible events to which the control may respond are selected from a predefined list, and for HyperCard, the events are freely assigned with a code. Typical events are shown in Table 1; these are very general events that almost all controls may receive from the user. There are, of course, specialized events that apply to particular controls—for example, the scroll-bar can receive a *scrollUp* or a *scrollDown* event.

Table 1
Typical Events for Programming Languages With Graphical User Interfaces

Input Method	Event	Object Receives Event When
Mouse	mouseDown	button is first down
	mouseUp	button has been released
	mouseClick	button used
	mouseDblClick	button double-clicked
Keyboard	keyDown	key is first down
	keyUp	key is released
	keyClick	key used
General	change	object changed
	gotFocus	received system focus
	lostFocus	lost system focus

When a control receives a user event, it is then up to the programmer to implement methods that handle the event and make the application respond in the appropriate manner. For example, when the user clicks the mouse on a button that says "Print Options," it will send the *mouseClick* event to the button, which, when it contains the correct code, may well bring up a dialogue box containing various user-definable options about printing. Although the highlighting of the button may be automatic, the implementation of the response is dependent on the programmer.

METHOD

What to Log

The logfile will record, with each user action, a descriptor string that contains the following information: (1) *who* is using the program; (2) *which* application are they using; (3) *when* are they using it; (4) *where* are they within the program; (5) *what* have they done; and (6) auxiliary data, as necessary.

Before defining how to log data, it must be clear exactly what information is required to classify the user's actions and the task that he or she is completing. In behavioral research, this may include several auxiliary items, to identify the particular experimental conditions under which the subject works.

Format

Between the individual data elements is an item *delimiter*, a characte⁻ that is used in a consistent manner to separate data elements. Typical delimiters include the space ( ), the slash (/), the colon (:), and the semicolon (;). It is important to note that some computer operating systems return date and time information with colon, slash, or space delimiters (e.g., 10:02:18, 18/9/97, 18 9 97). The actual format of the data and time is often set by the user in a preference file, so a consistent format cannot be guaranteed. The comma is also used by some sys-

```
data_item_1      data_item_2      data_item_3      data_item_4
Subject_1        Test_1           21/10/97         10:12:51          // Tab
Subject_1/Test_1/21/10/97/10:12:51                                   // Slash
Subject_1:Test_1:21/10/97:10:12:51                                   // Colon
```

Figure 1. Example of data items with tab, slash, and colon delimiters.

tems to denote numbers that are larger than one thousand (e.g., 12,345). It is, therefore, recommended that only the tab character (ASCII character 10) be used as the delimiter, since this ensures that individual data elements will remain separate. Figure 1 shows items separated with tab, slash, and colon delimiters and illustrates the merging of the data that can occur without an effective delimiter.

To record a logfile so that it is suitable for analysis with computer statistics packages, care must be taken with the actual format of the data. It is important that the data be recorded in the same order and be of similar data types. For example, always store the time followed by the current screen name followed by the user action. If a data element is optional or empty, there are two ways to record this: Store nothing, or use a marker to indicate no value. Some analysis packages accept the first method, where it will actually just be a record of two tab characters. For others, it must be explicitly recorded that it has a null value (e.g., using the actual word *null*). These null markers can later be replaced in a text editor, if the analysis package requires a particular character to represent no value.

The data to be recorded with each user action can be separated into two parts: *static* data and *dynamic* data. These are described below.

Static data. These are data that do not change while a single user is working with the application. When testing software under experimental conditions, these would be the subject's personal data and a designator for the current test. Figure 2 shows an example of static data.

Much of the personal data is recorded in *categories*; this is necessary in order to simplify analysis with statistics packages. For example: If there are 20 subjects, their element names may range from S1 ... S20; if there are two different tests, they may be called T1 and T2. It is safer to use an alphabetic character followed by a numeric character, as this makes the elements easier to differentiate than does simply a number, and it is often necessary so that the data can be analyzed by category. Although meaningful names, such as "Computer_Test_1," are easier to read and understand, the resulting increase in logfile size can make storage during the test more difficult and the later analysis slower, with some statistical packages.

Dynamic data. These are recorded data that change as the user interacts with the system; they provide a log of the user's actions. The main part of these data records what the user has done, where the user was within the system, and when he or she made that action (the what, where, and when of logfiles). Figure 3 shows an example of the dynamic data part.

The dynamic data contain the time of the action, often called the *time stamp*, which should contain at least two measures: the current time and the incremental time. The current time is the time, given by the system, when the event occurred. This is usually given in a format defined by the operating system. As was previously discussed, the actual format may change from machine to machine on the basis of user preferences. The incremental time contains the time difference between actions—that is, the time since the last action was made. If this is also recorded, it makes analyzing time data much easier, since this is a frequently required time measure. The processing time of the computer may also be recorded, to give an indication of how long the system took to process various user actions or even the actual logging functions

```
S1    43    M    19/9/97    T1    E1    J1
```

Where:

S1	subject number
43	age
M	gender (M/F)
19/9/97	date of the test
T1	test number

Additional static elements may be:

E1	experience category
J1	occupation category

Figure 2. Example of static log data.

```
10:12:34    9.43    Print_Mask    Button    Print    MO    Click    Left
```

Where:

10:12:34	current time
9.43	incremental time (seconds)
Print_Mask	name of current active window
Button	object type: button, text_field, menu etc.
Print	object name
MO	input method that fired the event: mouse, keyboard
Click	action: Click, DblClick, KeyPress, etc.
Left	control dependant data;
	for mouse : which button (Left, Middle, Right)
	for keyboard : actual key pressed

Figure 3. Example of dynamic log data.

themselves. This may be important when determining how long the user had to wait before receiving some form of feedback from the program.

How to Log

This involves three steps: (1) get the specific information from the control; (2) merge the static and dynamic data; and (3) update the logfile.

The most important aspect of logging event-driven applications is that all user's actions must be recorded. For this to be accomplished successfully, every object with which the user can interact must be able to record this interaction. Rather than writing many different routines, it is often better to make a general routine that can be called from each control. This general routine must supply the essential information that identifies this particular control from others, the type of interaction, and the results of this interaction.

As an example, the mouse is clicked on a button named "MyButton" on a form (a dialogue window) called "MyForm." The button may contain one of the following scripts:

```
For VB:          Private Sub MyButton_MouseUp
                 (Button%, Shift%, X!, Y!)
                    LogData "MyForm","But",
                 "MyButton","MO","mouseUp",Button%
                 End Sub

For HyperTalk:   on mouseUp
                    LogData (the long name of me),
                 "MO","mouseUp","NULL"
                 end mouseUp
```

Note that, in VB, the current position of the mouse (X! and Y!) and whether the Shift, Control, or Alt key (Shift%) is used are automatically provided. If these are required in the logfile, they could also be sent to the *LogData* function. Specifically for VB, the *ME* keyword provides a way to refer to the specific instance (including its properties) of the class where the code is running. The *ME* keyword is particularly useful for passing information about the currently executing instance of a class to a procedure in another code module. In HyperTalk, the command *the long name of me* returns a description of the object that contains this code, including its name, the type of object (button, field, etc.), the name of the current card, and the name of the current window. The procedure *LogData* must add to this information the static data and the time stamp.

The data from each user event can be simply appended to an open text file, or if this is too slow, all the data for a single user can be stored in memory and then written to the file when the test has finished. In HyperCard, data created when the test program is running can be stored in text fields that will keep this static text even when the program is exited and restarted. This can be very useful in the event of a system crash.

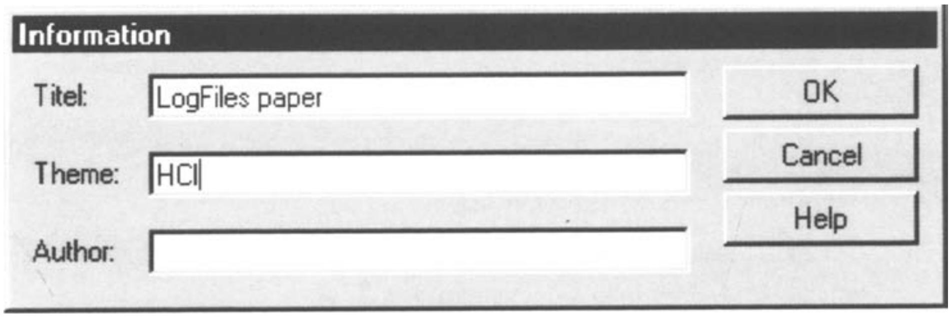
RESULTS

The data recorded in logfiles is best demonstrated with examples.

Example 1

This is a typical dialogue box for entering information about a document under the Windows 95 operating system, as is shown in Figure 4.

Table 2 shows that Subject 1 (S1), aged 43 and female (F), was completing Test 1 (T1) on September 10, 1997. At 13:23:12, the letter *H* was typed on the keyboard



Titel: LogFiles paper

Theme: HCI

Author:

Figure 4. Example of a dialogue box from the Windows 95 platform.

Table 2
Extract of Data Recorded to Logfile for Example 1

S1	43	F	10/9/97	T1	13:23:12	12.34	Info_Mask	txtF	Theme	KB	Key_Press	H
S1	43	F	10/9/97	T1	13:23:13	0.56	Info_Mask	txtF	Theme	KB	Key_Press	C
S1	43	F	10/9/97	T1	13:23:14	0.64	Info_Mask	txtF	Theme	KB	Key_Press	I

(KB) into a text field (txtF) called "Theme" in a window called "Info_Mask." It was entered 12.34 sec after the last entry. The letters *C* and *I* were then typed into the same field, after 0.56 and 0.64 sec, respectively.

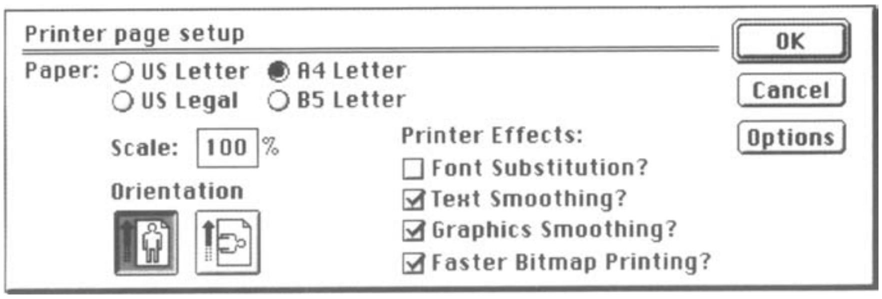
Example 2

This is a typical dialogue box from the MacOS platform, allowing the user to choose various printer page settings, as is shown in Figure 5.

Table 3 shows that Subject 2 (S2), aged 21 and male (M), was completing Test 2 (T2) on September 10, 1997. At 14:00:34, the left (Left) mouse (MO) button was clicked (Mouse_Click) on a button (But) called "A4" on a screen called "Setup_Mask." This happened 2.14 sec after the last action. On the same screen, 3.45 sec later, the button "Portrait" was clicked on, followed by the button "OK" after 2.09 sec.

CONCLUSIONS

Data logging is one of many human factors tools that can be used to determine how software is being used and has demonstrated its effectiveness in aiding the development of new systems (Good, 1985). Because of the purely objective nature of the data recorded, logfiles are unable to indicate a subject's satisfaction when using a program. When evaluating the usability of a program, this therefore requires that other methods also be employed. For behavioral experiments, the user's reaction times may be all that is required, and so a simple logfile may suffice.



Printer page setup

Paper: ○ US Letter ● A4 Letter ○ US Legal ○ B5 Letter

Scale: 100 %

Orientation

Printer Effects:
☐ Font Substitution?
☑ Text Smoothing?
☑ Graphics Smoothing?
☑ Faster Bitmap Printing?

Figure 5. Example of a dialogue box from the MacOS platform.

Table 3
Extract of Data Recorded to Logfile for Example 2

S2	21	M	10/9/97	T2	14:00:34	2.14	Setup_Mask	But	A4	MO	Mouse_Click	Left
S2	21	M	10/9/97	T2	14:00:38	3.45	Setup_Mask	But	Portrait	MO	Mouse_Click	Left
S2	21	M	10/9/97	T2	14:00:40	2.09	Setup_Mask	But	OK	MO	Mouse_Click	Left

For wider knowledge of a user's actions, logfiles can be most effectively used as an addition to other methods or as part of a combined method (Hietala, 1987).

As Hoiem and Sullivan (1994) state, "Experience has shown us that collecting data from many sources contributes to a more detailed yet broader picture of the human–computer interaction" (p. 169). That is probably the most important factor for software evaluation methods. Although the total amount of data collected is greater, the limitations of one type of data are overcome by the others.

Data logging, as presented in this paper, can be a useful tool for the behavioral scientist or software evaluator, but its use requires preplanning to ensure that it provides serviceable data. First, the software must be capable of generating the necessary events for all user actions on all interface elements. This involves adding code to the event-handling routines so that they call the necessary logging function. Under certain experimental conditions, only some actions may be of interest, but it is a better experimental practice to filter these from a complete record of all actions, rather than to work with a reduced set. Finally, implementing the logging function is simplified if it is introduced as the program is under development, rather than adding logging functions to the finished program.

REFERENCES

Beizer, B. (1990). *Software testing techniques* (2nd ed.). New York: Van Nostrand Reinhold.

Good, M. (1985, April). *The use of logging data in the design of a new text editor.* Paper presented at Human Factors in Computing Systems II: CHI '85, San Francisco.

Henderson, R. D., Smith, M. C., Podd, J., & Varela-Alvarez, H. (1995). A comparison of the four prominent user-based methods for evaluating the usability of computer software. *Ergonomics*, **38**, 2030-2044.

Hetzel, W. (1985). *The complete guide to software testing.* London: Collins.

Hietala, P. (1987). Combining logging, playback and verbal protocol: A method for analysing and evaluating interface systems. In J. Rasmussen & P. Zunde (Eds.), *Empirical foundations of information and software science III* (pp. 99-108). New York: Plenum.

Hoiem, D. E., & Sullivan, K. D. (1994). Designing and using integrated data collection and analysis tools: Challenges and considerations. *Behavior & Information Technology*, **13**, 160-170.

Jeffries, R., Miller, J. R., Wharton, C., & Uyeda, K. M. (1991, April). *User interface evaluation in the real world: A comparison of four techniques.* Paper presented at Reaching Through Technology: CHI '91, New Orleans.

Nesi, P., & Serra, A. (1995). A noninvasive object-oriented tool for software testing. *Software Quality Journal*, **4**, 155-174.

Palya, W. L., & Walter, D. E. (1993). A powerful, inexpensive experiment controller or IBM PC interface and experiment control language. *Behavior Research Methods, Instruments, & Computers*, **25**, 127-136.

Siochi, A. C., & Ehrich, R. W. (1991). Computer analysis of user interface based on repetition in transcripts of user sessions. *ACM Transactions on Information Systems*, **9**, 309-335.

Siochi, A. C., & Hix, D. (1991, April). *A study of computer-supported user interface evaluation maximal repeating pattern analysis.* Paper presented at Reaching Through Technology: CHI '91, New Orleans.

Smith, S. L., & Mosier, J. N. (1986). *Guidelines for designing user interface software* (Report AD-A177 198). Bedford, MA: National Technical Information Service, MITRE.

Yamagishi, N., & Azuma, M. (1987, August). *Experiments on human–computer evaluation.* Paper presented at Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems, Second International Conference on Human-Computer Interaction, Honolulu.

(Manuscript received November 20, 1997;
revision accepted for publication April 13, 1998.)