# Inputlog: A logging tool for the research of writing processes

*Mariëlle Leijten & Luuk Van Waes*
*University of Antwerp, Belgium*

The use of computers as writing instruments has not only had a profound effect on the writing practice and the attitudes towards writing, it has also created new possibilities for writing research. In the field of cognitive writing research especially, keystroke logging programs have become very popular. In this paper we describe a new logging program, called Inputlog. The program consists of three modules: (1) a data collection module that registers on line writing processes on a very detailed level; (2) a data analysis module that offers basic and more advanced statistical analyses (e.g. text and pause analysis); (3) a play module that enables researchers to review the writing session. The technical and functional description of Inputlog is complemented by two examples of research studies in which the program was used. We wind up the paper with a preview of the plans for further developments.

Keywords: keystroke logging, registration tool, writing processes, pauses, writing modes, text analysis, pause analysis, writing observation, cognitive processes

## 1. Introduction

In current writing research the description of cognitive processes is one of the most important domains. As most writers nowadays produce nearly all of their texts on a word processor, the computer has not only become the major writing instrument, its use as a research tool has also increased. The computer enables researchers to collect detailed information about the writing process that was hardly accessible before. Or, as Spelman Miller and Sullivan (to appear) state in the introduction of their anthology on key-stroke logging: "As an observational tool, keystroke logging offers the opportunity to capture details of the activity of writing, not only for the purposes of the linguistic, textual and cognitive study of writing, but also for the broader applications concerning the development of language learning, literacy, and language pedagogy." (p. 1).

Nowadays, researchers make frequent use of 'keystroke logging tools' to describe writing processes (see Sullivan and Lindgren (to appear) for a review). These logging programs enable researchers to exactly register and accurately reconstruct the writing processes of writers that compose texts at the computer. The basic concept of the different logging tools that have been developed is more or less comparable. First, the keystroke logging tools register all keystrokes and mouse movements. During the writing process these basic data are stored for later processing. This continuous data storage does not interfere with the normal usage of the computer, creating an ecologically valid research context. At a later stage, the logged data can be made available for further analysis, either within the program environment itself or as exported data in statistical programs such as SPSS or SAS. Depending on the research question, researchers can choose to analyze different aspects of the writing process and the writing behavior by combining, for instance, temporal data (e.g. time stamps or pauses) with process data (keystrokes or mouse movements). The data collection (and processing) can be performed much faster and more accurately by means of the computer, than it could ever be done manually.

Keystroke logging can be applied to various fields of writing research. It enables researchers, for instance, to describe and compare the writing processes of novice and expert writers in professional business contexts, to analyze the writing behavior of writers with writing, reading or learning difficulties or to compare writing strategies in first and second language writing processes. Sullivan and Lindgren (2006) present several studies in their book in more detail: fundamental research on keystroke logging as a research instrument (Stromqvist et al.); writing research on temporal aspects of the writing process (Wengelin; Spelman Miller), and writing research on revising behavior (Lindgren). The above-mentioned studies do not provide a conclusive list of keystroke logging research but they

are rather indicative of what can be studied by logging writing processes. On the one hand that is the logging tool on itself and on the other hand that are studies that can help to develop more fundamental theories about e.g. formulation processes or revision processes.

In this article we describe Inputlog, a logging tool for writing process research developed for Windows environments. First, we give a short overview of the most important logging tools that are currently available for research purposes. We describe their main characteristics and their influence on the development of Inputlog. Then we present a more detailed description of the technical and functional characteristics of Inputlog. In the last sections we illustrate the use of Inputlog in two exemplary research studies and preview the further developments of the program.

## 2. Overview of logging tools

For the development of Inputlog we were able to fall back on the functionality of two existing programs: JEdit and Trace-it on the one hand (Eklundh 1994, Eklundh & Kollberg 1992; Severinsson Eklundh 1996; Severinsson Eklundh & Kollberg 1996, 2003; Kollberg 1998), and ScriptLog on the other hand (Strömqvist & Malmsten 1997). JEdit and Trace-it are only suitable for Macintosh personal computers. JEdit only logs data in an in-house developed limited word processor. ScriptLog also mainly logs in a limited word processor that was developed for research purposes (i.c. mainly writing experiments with young children). Trace-it features an extended revision module, while ScriptLog combines logging data with recorded eye-tracking data.

### 2.1. JEdit

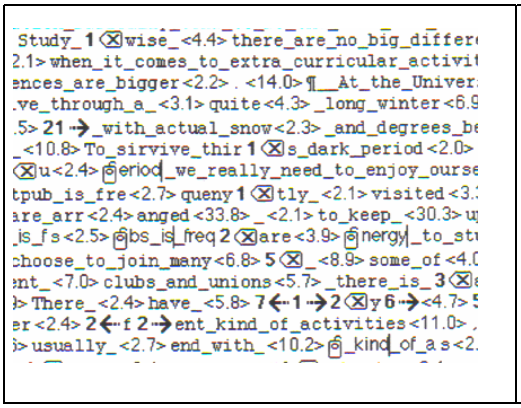
JEdit[1] is a logging word processor that has been developed for Macintosh. JEdit supports some of the most common text editing functions (e.g. cut, paste, undo, redo, text formatting). Furthermore, it provides basic statistics about writing sessions, such as information about pauses, commands used, input devices used, etc.

JEdit logs a writing session in an icon-based log file format (see Figure 1). This log can be exported to a file in a so-called *MID*-format. MID - an acronym for 'Movement, Insertion and Deletion' - is an editor independent format that summarizes every elementary operation in the writing process in these three actions and tags them with a unique time stamp so that the writing session can be analyzed using the S-notation (Kollberg, 1998). In the S-notation the final text of the logged writing session is represented in a semi-linear way, including all revisions (see Figure 1).

Every revision the writer has performed is represented in the order of occurrence while retaining the internal structure. In sum, the exact history of the text is represented, instead of the purely linear, chronological representation of the logging file.

The data in MID-format can be read by Trace-it, an interactive computer program specifically designed to support the analysis of revisions. Based on the S-notation, the program replays the text on the screen and mainly focuses on the revision process. The interface is easy to use and the text representation is system-independent. Therefore, it can be used to study revision strategies of writers using different computers or word processors. The Trace-it environment supports various types of interactive analyses and also features a replay mode of an entire writing session.

[1] For more information: http://www.nada.kth.se/iplab/trace-it



Example of text in S-notation:

I am writing a {short}1text.|[1] It will[ probably[2]|[3] be revised [ somewhat][3] later.|[2] Now [I am|[4]][4]it is finished.

The final text produced in this example reads as follows:

I am writing a short text. It will be revised later. Now it is finished.

The S-notation uses the following symbols:

| | |
|---|---|
| $\|^i$ | The break with sequential number $^i$ |
| {inserted text}$^i$ | An insertion following break # $^i$ |
| [deleted text]$^i$ | A deletion following break # $^i$ |

Figure 1 *Example of icon-based log file JEdit and text in S-notation*

Trace-it shows the text in S-notation in one window, and offers the researcher the possibility to navigate between the revisions. In an additional window, the plain text is shown, and the user can replay the writing session revision by revision, forwards or backwards (see Appendix 1). Statistics of the writing session can also be obtained, either as a summary or as a detailed description.

## 2.2. ScriptLog

Like JEdit, ScriptLog[2] can log any writing activities that take place on a computer: every keyboard and mouse action, the screen position of these events and their temporal distribution. Initially, ScriptLog only logged data in a limited and custom-designed word processor. In more recent releases, the developers also offer limited logging for commercial word processors (keystrokes but no mouse movements). ScriptLog offers a wide range of low level analyses that can be generated from the data, e.g. pauses, strings, transitions and deletions.

A distinguishing characteristic of ScriptLog is that its recordings can be combined with data of an eye tracker. Furthermore, ScriptLog provides facilities for designing writing experiments using elicitation instruments (e.g. pictures and time stimuli).

# 3. Characteristics of Inputlog

As mentioned above, most logging-tools are either developed for a specific computing environment, or not adequately adapted to the current Windows environment. As such, they cannot be used for writing studies in which 'natural' writing and computer environments employ commercial word processors (e.g. MSWord or WordPerfect). This discrepancy has been the main reason for deciding to develop a new logging tool, i.e. Inputlog.

Another impetus for the development of Inputlog has been a study[3] on the influence of speech recognition on the writing process. For this study we observed the writing processes of twenty participants who used speech recognition software during their day-to-day work in their professional business contexts (five observations of approximately half an hour per participant). Because it was not possible to register keyboard input in combination with speech mode data with any of the current logging tools, the process data had to be analyzed manually. To collect the writing process data, or should we say speech recognition data, we combined two digital observation instruments: a digital screen cam (i.c. Camtasia) and a digital sound recorder (i.c. Quickrecord). The study showed that the chosen observation instruments and analyzing methods did enable us to analyze and describe the

[2] For more information: http://www.scriptlog.net

[3] The study on the influence of the use of speech recognition on the writing process was part of an NOI research project (Research grant of the University of Antwerp 2000-2002 and 2002-2004).

specific speech recognition writing processes (Leijten & Van Waes 2003; 2005) even though the data analyses were very time-consuming.

As a result, we started to develop Inputlog in 2003. This program was designed to make analyses of several characteristics of writing processes faster and more accurate. In short, Inputlog is a logging tool that enables researchers to:

- Record the data of a writing session in MSWord.
- Generate data files for statistical, text, pause and mode analyses.
- Play the recorded session at different speeds.
- Capture the input of dictation processes using speech recognition software (i.e. Dragon Naturally Speaking, planned implementation, see section 7).

The most distinguishing characteristic of Inputlog to date is its word processor independent functionality. Contrary to Trace-it and ScriptLog, Inputlog registers every keystroke and mouse movement independently of the word processor used. Inputlog is designed to log and anlyze writing data produced in MSWord. However, the program also logs keyboard and mouse actions in other Windows based programs.[4] In other words, not only writing processes as such can be observed with Inputlog, but also basic processes like consulting websites or programming in any Windows based language can be observed.

Apart from logging writing sessions, Inputlog offers researchers the possibility to analyze writing processes from different perspectives. The output data of the writing process registration is saved in a source file, a so-called IDF-file. In this source file every action of a writing session is saved. Figure 2 visualizes the flow of the program[5].

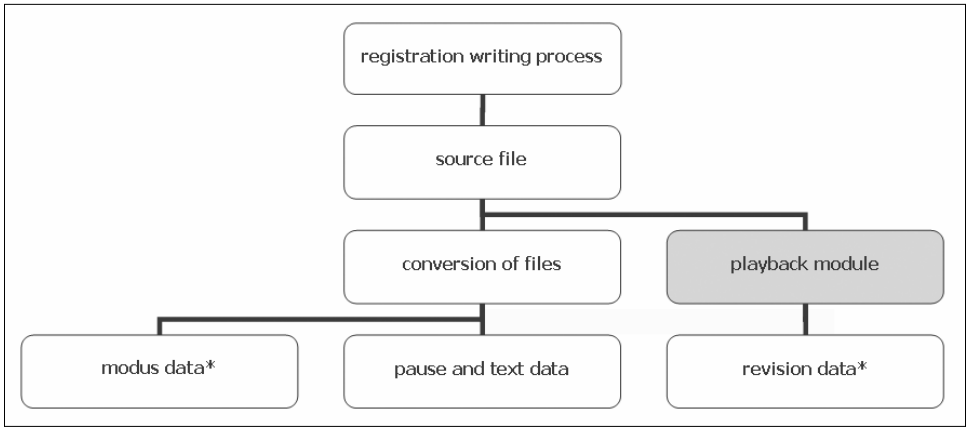



Figure 2: Basic flow of the logging and analysis conversion in Inputlog

The source file data can be used either for the conversion of files, or as an input for the playback module. The conversion of files 'translates' the data in the source file to separate data files that enable researchers to analyze the modus data, the pause data and the text data on a more aggregated level. The playback module enables researchers to replay the writing session exactly as it was registered, or sped up to the researcher's preference. Furthermore, the playback module is also used as an intermediate facilitator for the revision analysis.
We have developed an interface in which researchers can select the analyses that have to be performed for a specific source file. This user-friendly interface allows researchers to use Inputlog in

[4] Researchers should keep in mind that for certain analyses (e.g. analyses on sentence or paragraph level) only the data logged in MSWord can be used. For this kind of analyses the basic data are interpreted with a set of algorithms based on MSWord.

[5] The data with an asterisk are not active in Inputlog version 1.4.

function of their research needs. This basic functionality is described in section 4. In section 3 we first give some more information on the technical background of Inputlog.

# 4. Technical description

In this section we describe Inputlog from a more technical perspective. Technical terminology will be further explained in the glossary (see Appendix 2). Inputlog captures input data at a level before they are converted to screen information. It captures:

- scancodes of keystrokes
- mouse activities (clicks, movements, location)
- time stamps of all input events

These data are stored in so-called IDF-files (Inputlog Data Files) that are converted to different output files afterwards, preparing the rough data for qualitative and quantitative analyses (cf. functional description in section 5).

## 4.1. Programming language

Inputlog is programmed in Visual C++ and the interface is programmed in Visual Basic. We have opted for C++ as a programming language because it is object oriented and because of its processing speed. Each part of Inputlog is programmed in different classes (see **Fout! Verwijzingsbron niet gevonden.**). This enables us to easily update and debug the current analyses, and to further extend the functionality of the program. The speed of the program is very important because the resident logging program may not interfere with the usage of the word processor, and consequently, it may not hinder the writer during his or her writing process in any way. Inputlog runs on a PC with only one CPU and its use of system resources is quite limited, contrary to, for example, Java.

For the logging of the writing process Inputlog uses two Windows Hooks: *Journalrecord* and *Journalplayback*. *Journalrecord* registers every keystroke and mouse operation (movement and click) together with the corresponding time stamps. The *Journalplayback* is the DLL-file that supports the playback module.

## 4.2. Structure of Inputlog

Figure 3 shows the structure of Inputlog, or the so-called 'system topology'. Inputlog consists of four subsystems: storage, replay, logging and analysis. The storage system activates the classes' session identification and processes the logging data. The session identification saves the relevant data to identify and name the session. The logging data saves all the logging data in a designed algorithm that is stored in an array list.

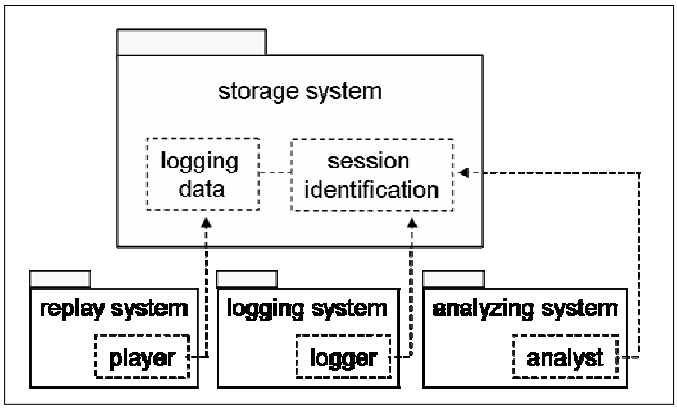



Figure 3: System topology of Inputlog

The storage system has three subordinated systems: replay, logging and analysis. These three subsystems are all dependent on the storage system.

- The replay system needs logged data to replay a writing session.
- The logging system needs to provide information to the session information.
- The analysis system needs information about the session identification to confirm and name the correct writing session.

The session identification can run any type of analysis by initializing the right class (e.g. PauseAnalyst initializes the pause analysis). The classes' player and logger are so-called 'singletons', viz. only one of them can be active in the system. Consequently, Inputlog always needs to perform a consecutive analysis to verify that only one of these two subsystems is active: the logger can not log while playing and vice versa.

The three subsystems can function independently. The GUI interacts with the session identification to play, log or analyze a logging session.

## 4.3. Structure of IDF-files

Each IDF-file contains the data of one logging session. In this file the logging data are combined with the session identification data (see section three). The structure of the IDF-file is shown in Figure 4.

<table>
<tr><th>bytes</th><th>meaning</th></tr>
<tr><td>4</td><td>starting time of the log in milliseconds</td></tr>
<tr><td>14 * 4</td><td>14 digits that indicate the size (viz. number of signs) of the session variables<br>(6 values for the defined variables, 4 for the user defined variables and 4 for the undefined values)</td></tr>
<tr><td>14 * #</td><td>14 session variables * number of the size that has already been read</td></tr>
<tr><td># * 20</td><td>serial number of keyboard and/or mouse event (type Eventmsg)<br><br>Each event that is logged by Inputlog creates a log event of 20 bytes. An Eventmsg is:<br><br>
<table>
<tr><th>bytes</th><th>meaning</th></tr>
<tr><td>4</td><td>message type (key in, key out, mousemovement, left mouse button in, left mouse button out, etc.)</td></tr>
<tr><td>4</td><td>paramL<br>for keystrokes: virtual keycode<br>for mouse events: x-value of location of the mouse</td></tr>
<tr><td>4</td><td>paramH<br>for keystrokes: scancode<br>for mouse events: y-value of location of the mouse</td></tr>
<tr><td>4</td><td>timestamp in miliseconds starting from the beginning of the log<br>(time event – starting time of computer) * starting time of the log</td></tr>
<tr><td>4</td><td>handle to active window</td></tr>
</table>
</td></tr>
</table>

Figure 4: Structure of the IDF-file

## 4.4. Conversion of files

The session identification information and the logging data are converted to a Microsoft Excel sheet or an HTML-file. The data files can also be converted to SPSS, or any other statistical program. Because the data are prestructured to facilitate further statistical analyses, it is important to define labels and variables before starting the writing session. Inputlog provides the possibility to attribute 6 predefined (participant, age, sex, session, group, experience) and 4 extra user-defined variables.

Empty columns in the session identification, viz. undefined variables, are automatically assigned the missing code "99". This eases the process of filtering empty columns that are not needed for further analysis. Besides, in Microsoft Excel, the first column is automatically generated. This enables researchers to assign a unique number to every single variable or row.

## 4.5. Program settings

Inputlog is dependent on some basic settings that are particular to the computer configuration that is used to log the writing session. We shortly discuss three main configuration elements: screen settings, keyboard lay-out, and the memory allocation.

### 4.5.1. Screen settings

As stated before, Inputlog replicates the writing session exactly as it is logged. Therefore, it is important that the computer settings - both screen and program settings (e.g. toolbars, language settings, personal dictionaries etc.) - are exactly the same when replaying a writing session as when the writing session was recorded. A short example is given to illustrate this issue. A writing session is recorded with only the standard toolbar active. However, between the logging session and the moment the observation session is replayed, the formatting and reviewing toolbar are added to the working environment which results in a different positioning of certain (graphic) elements on the screen in comparison to the previous screen outline. Consequently, because the replay uses the graphic xy-postion of mouse clicks, the cursor might select a different icon or item than it did at the time the session was recorded. Figure 5 shows, for instance, that during a replay with different screen settings, instead of changing the font into size 12, the paste-icon was selected. The result is that the continuity of the writing process reconstruction is severely disturbed. Therefore, researchers are recommended to keep the settings of the computer screen exactly the same between logging and replaying.

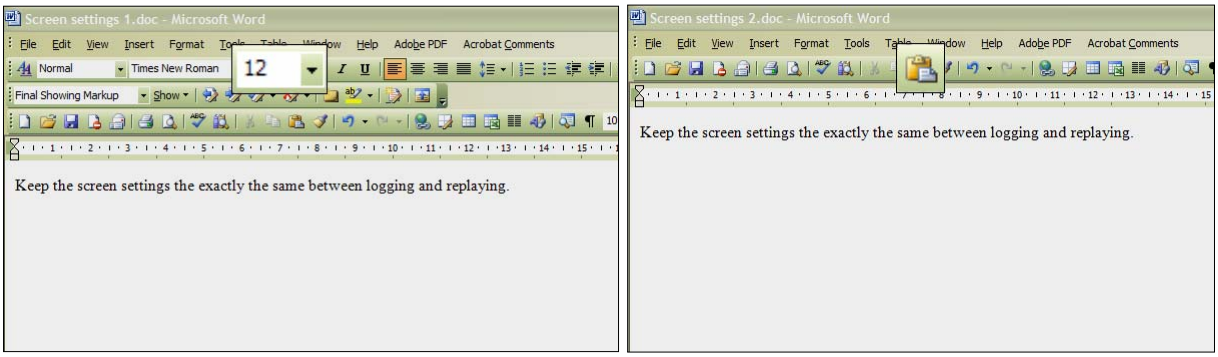



Figure 5: Illustration of a replay with different screen settings

Note that this particular point of interest should only be taken into account when using the replay function of Inputlog. The generation of the different analyses[6] is not disturbed by different settings.

### 4.5.2. Keyboard layout

A second configuration aspect that can influence the logging of a writing session are the keyboard layouts of the computer used during the logging session. As a result of the amount of different keyboard layouts, Inputlog has to detect the correct layout of the keyboard used. For instance, if a writing session is logged with QWERTY-settings and the actual layout of the keyboard is AZERTY, the following sentence will be represented incorrectly in the logging output:

[6] Because the revision analysis is also based on the replay function, the settings should also be consistent for this analysis. However, in Inputlog 1.4 this function is not yet activated.

Typed sentence: *This is a short writing session in Inputlog.*
Logged sentence: *This is **q** short **z**riting session in Inputlog**.***

To avoid this problem Inputlog is programmed to detect the correct hard-coded keyboard layouts for the lowercase characters and reads the connected Windows settings for the uppercase characters. At this moment, 33 different keyboard layouts are predefined in Inputlog 1.4 (see the program's help-file for a detailed list).

## 5. Functional description

The previous section described the technical background of Inputlog. In this section we shortly explain the basic functionality of the program and its interface. The interface of Inputlog consists of an entry screen and 4 different tab pages: record, generate, play and help (see Figure 6). The help file is a standard html-file that can also be consulted without starting Inputlog itself. For more detailed information about the use of the program we refer to this file.

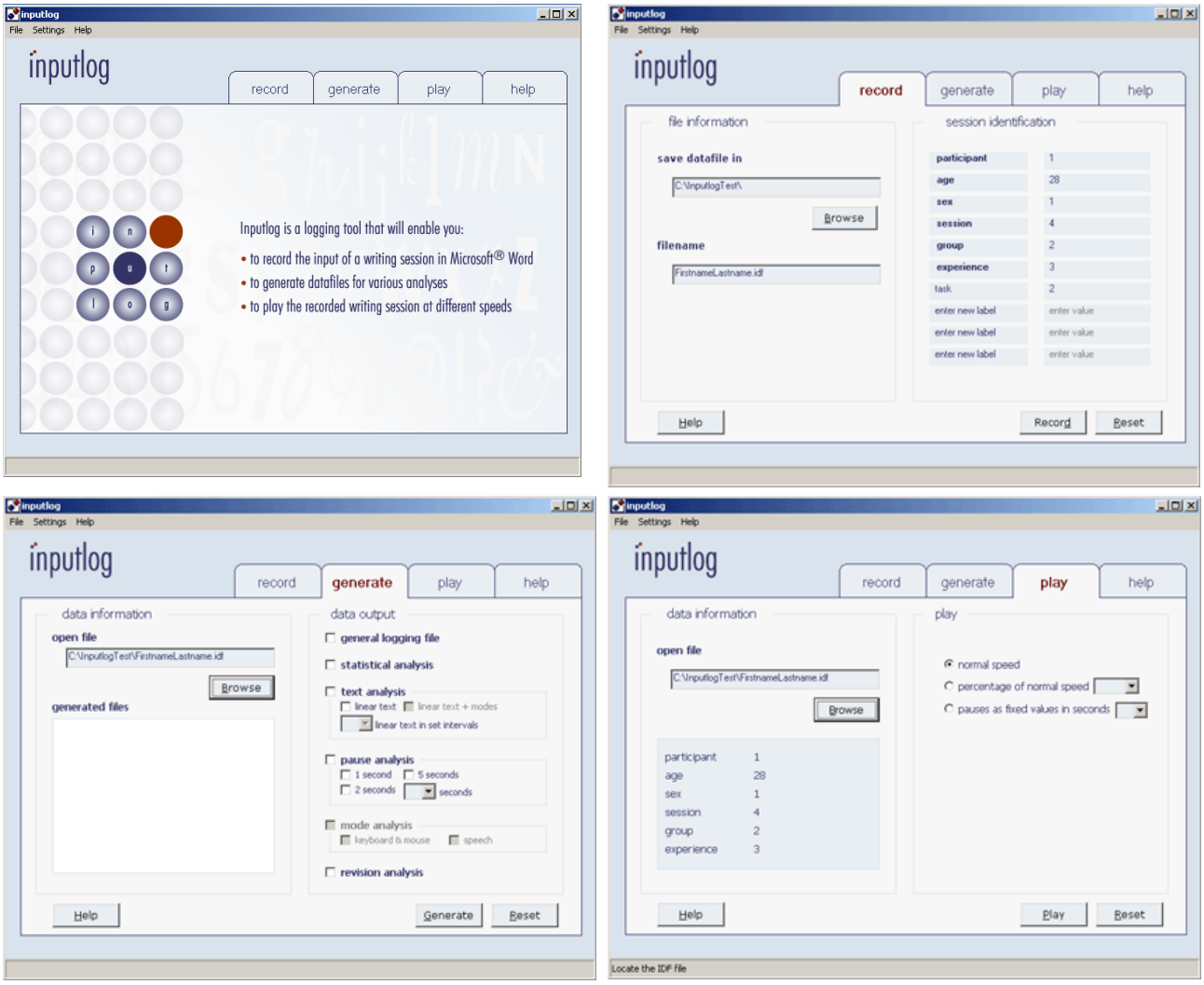



Figure 6: the four main tab pages of the Inputlog interface

## 5.1. Entry screen

Inputlog starts with an entry screen that provides the user with a short overview of the program. The user can opt to start a new logging session, he or she can generate files from an existing logging file or he or she can choose to replay a logged session. The different tabs give the user direct access to the different functions. By using the main menu items at the top of the screen - File, Settings and Help - basic file operations can be activated, program settings can be changed and the help file can be accessed.

## 5.2. Record

By selecting the record tab, users can start a new logging session. However, before a new session is started the researcher can first specify the *file information* and the *identification data* for a specific session. This information will be included in all the analysis files that will be generated based on the session source file, facilitating the identification of each writing session.

In the file information, users can indicate where they want to save the logging session on their computer hard or network drive, and they can enter the unique filename for the source file, e.g. FirstnameLastname1(= Firstname participant + Lastname participant + Number of session). The file that will be generated in the recording session has the extension *.idf, which is added automatically. This file will be used as an input for the generate and play functions (cf. infra).

Each logging session can be identified by a maximum of ten variables (6 predefined and 4 user defined variables). These variables should enable the researcher to identify a writing session in detail.

## 5.3. Generate

The generate tab opens a window showing the different analysis options. In this part of the program, analysis files can be generated on the basis of a source file that was recorded in a previous logging session. In other words, any IDF file can be opened at any time to generate data output files for specific analyses. The generate window exists of three sections. In the data information section, users can specify the IDF file from which they would like to generate the analyses. In the data output section the researcher can specify which output needs to be generated. In the section below the data information, a list of all the files to be generated, is displayed.

Inputlog 1.4 offers 4 different data analyses:

1. **general logging file**: a spreadsheet with a basic log file of the writing session in which every line represents an input action (letter, function, mouse click or movement); for every input action the session information is stored together with an identification of the input, the time stamp, the pausing time that followed it, and – for a mouse operation - the xy-value of the screen position (for an overview see Appendix 4).
2. **statistical analysis**: a spreadsheet with basic statistical information on the writing session such as the session information, some basic data about the written text (product and process), pausing behavior and the use of the different writing modes.
3. **text analysis**: a plain linear text in HTML format with the complete linear production of the text including mouse movements and other activities; extra options allow for the production of a linear output in which the writing activities are divided into periods (fixed time periods of x seconds, free to choose) or intervals (fixed number of intervals in which the writing process is to be divided, free to choose).
4. **pause analysis**: a spreadsheet with analyses of every non-scribal period; the threshold for the pauses can be set to 1, 2 or 5 seconds as a standard or to any user defined level.

Two other analyses are under construction at this moment:

5. **mode analysis**: a spreadsheet with information about the distribution of the writing modes (keyboard, mouse, speech technology) that were used as an input device during the writing session.
6. **revision analysis**: a spreadsheet with a basic analysis of the number, the level and the kind of revision that has taken place during the writing session.

Inputlog takes some time to generate the requested files; the progress is shown in the 'generated file' section. The different files are all placed in the same folder as the source file of the selected writing session (cf. 3.2 record). This allows users to keep track of the recordings and analyses per writing session. The Excel files of the statistical and pause analysis consist of two parts (worksheets): a *full analysis* and an *overview* or *summary* of the writing session. The full analyses enable researchers to merge different files in a statistical program and to run further statistical analyses on the data. The overview on the other hand provides a short summary of the data. In addition, Inputlog creates for each level of pause analysis another worksheet in Microsoft Excel (e.g. pause-analysis-1-second and pause-analysis-5-seconds; see Appendix 3 for further information).

## 5.4. Play

A recorded writing session can be replayed using Inputlog. Again, the IDF file is used as a source file for the replay. To verify the information labels of the file that is selected for a play back session, all defined variables of the session identification appear in the dialog box on the left side of the screen. The writing session can be replayed at different speeds. It can be played back exactly as it was recorded (in real time): this is an exact reproduction of the recording of the session. Another option is that users select a percentage of the real time speed. However, we would like to restrict this option to a maximum of 120% of the original speed. Otherwise the program may have difficulties in processing and performing certain actions that require extensive memory access. The final option *'pauses at a fixed value'* enables researchers to assign a fixed value, for example 0.1 seconds, to every pause (non-scribal activity). This allows users to view a writing session without long interruptions or pauses.

## 5.5. Help file

The help tab contains a structured manual of the program. The help tab is a combination of a short instruction of Inputlog functionality and a program description. The main part of the help file consists of three chapters, each corresponding to the three basic functions of the software tool: record, generate and play. The help on each of these functions is also directly available via the help button on the associated tabs.

# 6. Usage

To illustrate the possible usage of Inputlog we shortly discuss two fragments of writing processes taken from experiments in which we have used the program as a logging device. The first experiment concerned a more technical writing experiment that focused on the working memory requirements necessary for error detection in the 'text produced so far' (Leijten, Ransdell & Van Waes, in preparation); the second example is taken from a case analysis of a writer producing a bad news letter.

## 6.1. The text produced so far and the use of working memory

An experimental setting was set up to assess the memory load during the 'text produced so far' interaction in the text production phase. The design included the most frequently occurring error types found in a case study of professional writers that were using speech recognition for the first time to write business texts (Leijten & Van Waes, 2005). In the example at hand, we selected two text fragments in which a sentence with large speech recognition errors is corrected. The errors that occurred in the text produced so far were considered as major because the number of characters that

differed from the intended text was more than two. Besides, the selected errors could only occur in speech recognition (see example Figure 7).

The task in the experiment consisted of a set of sentences that were presented to the participants to provide a new context (see Figure 7; 1). After every sentence the participants had to click the "ok" button, to indicate that they had finished reading the sentence. A sub clause of the previous sentence was then presented as text produced so far (TPSF) in a subordinate causal structure (2), and the participants were prompted to complete the sentence (3). In Figure 7 an example of a correct and an incorrect sentence is given.

**Correct**

1. Because the height was not indicated, the pick-up truck drove by the underpass.
2. The pick-up truck drove by the underpass,
3. because the height was not indicated.

**Incorrect**

1. Because the height was not indicated, the pick-up truck drove by the underpass.
2. The **picot trug** drove by the **underparts**.
3. because the height was not indicated.

Dutch "De **heeft week** reed onder de **brief** door, want de hoogte was niet aangegeven."

Figure 7: Examples of correct and incorrect sentences in the TPSF experiment

The participants could either prefer to correct the errors first and then complete the sentence, or to complete the sentence first and then correct the error. In casu, for this sentence 83% of the participants preferred to complete the sentence first and correct the error afterwards. Figure 8 shows the linear output of the writing session of two participants that used another writing strategy (see Appendix 5 for linear output abbreviations). Pauses longer than 1 second are included in the output; texts are in Dutch.

| **sentence completion → error correction (83%)** | **error correction → sentence completion (17%)** |
|---|---|
| [MwC.910,701-391,348] [McLeft] | **{2.66}** [McLeft] [MwC.910,711-149,347] |
| de hoogte was niet aangegeven **{2.19}** | [McLeft] *[MwC.149,347-375,355]* |
| [MwC.491,352-145,346] [McLeft] | [BS 9] |
| [BS 9] | ftruck **{1.6}** [MwC.372,357-269,340] [Mselect] |
| eftruck **{1.27}** | [MwC.270,340-342,341] |
| [RIGHT 17] [DEL 4] | [BS 4] |
| ug **<1.17>** | ug **<1.81>** [RIGHT 15] |
| [MwC.165,335-933,699] [McLeft] | de hoogte stond niet aangegeven **<1.25>** |
| | [MwC.348,338-884,695] [McLeft] |

Figure 8: Example of linear output of a text fragment generated by Inputlog

In these examples of two short writing fragments, different writing strategies can be distinguished. The first writer prefers to continue completing the text first, before correcting the mistake in the TPSF. He positions his cursor after the first segment (TPSF; cf. xy-value of left mouse click) – without a significant previous pause – and then completes the sentence. After the sentence is completed he pauses for 2.19 seconds. He then positions the mouse after the incorrect word and deletes it by using the backspace key. Next, he navigates through the text by pressing the right arrows key and deletes the second error. Finally, he has a short pause before he continues to the next sentence. The logging of the writing session of the second participant reveals a different pattern. He pauses before he starts writing and then positions the cursor behind the error to correct this first. After correcting the second error he completes the sentence.

In the analysis of the research data presented here, we were mainly interested in a description of the interaction with the text produced so far. Differences in writing strategies can be related to both individual differences and to error characteristics in the TPSF. This short example illustrates how the detailed process information that is generated by Inputlog provides a basis for analyzing writing strategies that are no longer visible in the final written product.

## 6.2. Pausing and revision behavior in writing bad news letters

In Figure 9 we show the linear text representation of a writer producing a bad news letter in which he declines an offer to deliver a keynote address for an international conference. We were especially interested in the process characteristics of the writing episode that concerned the wording of the bad news itself. As we know from the literature on this issue, the strategic considerations to make the bad news as acceptable as possible for the reader are crucial in the perception of the message (and may also determine the future interpersonal relation with the reader).

The writer needed about ten minutes to finish the letter of about 130 words. The replay function of Inputlog was used as a stimulus for a retrospective thinking aloud protocol. Figure 9 shows the sequential linear output (periods of 30 sec) of a fragment that illustrates the writer's strategic considerations at the beginning of the second paragraph.

| | |
|---|---|
| 150 sec | {8.44} Presenting a {1.33} paper to {2.36} this group {3.66} [CTRL+LEFT 1] {1.02} quality {1.8} [END] {1.03} deserves a {1.53} thor |
| 180 sec | ough {1.39} and {2.42}time {1.19} [BS 4] {1.66} comprehensive effort {2.13}.{3.61}<br>Of course, [BS 12]. {1.02} Obviously {2.90} |
| 210 sec | , such an effort {2.03} requires {5.55} a lot of [BS 8] {1.05} considerable [BS 12] time. {4.84} |
| 240 sec | However, {1.34} my schedule {2.66} [CTRL+LEFT 4] {1.28} [DEL 10] M [END] {2.38} is fully comm{1.77}itted to a |
| 270 sec | writing project {2.30} {7.17} [CTRL+LEFT 7] {2.16} |

Figure 9: Linear output of two fragments of a writing process in which a bad news letter was produced

The fragment shows a very fragmented and staccato writing process that starts off with a long pause of more than 8 seconds in the beginning of the second paragraph (announcement of the refusion). In the production of about 30 words, there were 28 pauses longer than one second, which is substantially more than in the previous period when the introductory context of the letter was written. About half of the time in this fragment is used for pausing, showing the time attributed to careful and strategic formulation. An interesting example of such a strategic consideration is the revision that takes place after four minutes (240 sec segment). The participant starts the sentence with 'However', but two words later he rereads the beginning of the sentence and realizes that this contrastive connective announces the bad news in a too early stage of the paragraph. Therefore, he decides to delete the connective to neutralize the context of this argumentative sentence.

Again, this example shows that process logging enables researchers to analyse writing processes from different perspectives enriching possible interpretation based on text analysis. Other observation methods, like recording (retrospective) thinking aloud protocols, might complement the acquired data.

# 7. Further development

To facilitate a broad usage of Inputlog, the program is put at the disposal of the research community for free, provided that reference is made to this paper in the publication of the study. The users' feedback is very important for the evaluation and further development of Inputlog.

We have identified four important niches that may increase the applicability of Inputlog, especially in the domain of writing process research. In the near future we would like to further develop the following (in order of priority):

1. a module for logging speech recognition events
2. a module for revision analysis
3. a module for progression analyses (basic and extended)
4. a module for integration with Morae (a macro oriented observation tool developed by Techsmith[7]).

In addition to these developments, we will pay special attention to the further development and optimization of the existing modules. Furthermore, the compatibility with different versions of the Window operating systems and the Office environment will require constant attention. Finally, the integration of our logging data with data from eye tracking observations is also on the agenda.

## 7.1. Speech recognition

Inputlog version 1.4 logs keystrokes and mouse movements. In the next version we would also like to log the input produced by speech technology, i.e. the speech recognition software Dragon Naturally Speaking. To facilitate the integration, Scansoft[8] added a new API in their latest professional version of Dragon Naturally Speaking 8.0 which enables us to integrate the dictated text with the data logged by Inputlog.

The implementation of speech recognition will stimulate research on the effect of this new technology on the writing process (of both professional writers and writers with learning disabilities). Moreover, we would also like to explore the logging possibilities of speech recognition to simultaneously transcribe thinking-aloud protocols and/or retrospective interviews.

## 7.2. Revision analysis

Work on the revision analysis for Inputlog has already been started, but because of its complexity it is very time consuming to further stabilize and extend the analysis. In the revision analysis we would like to produce an output analysis in which different characteristics of in-process revisions are described, e.g. the number of revisions, type of revisions, level of revisions, number of words and characters involved in the revision operation, as well as the location of the revisions in relation to the point of utterance. To define revisions we have developed an algorithm and a set of rules. The revision analysis first of all defines critical events in the writing process that might be linked to a revision and then evaluates these instances by comparing the operations in the isolated writing episode to the revision rules in the algorithm. Inputlog successively analyses the beginning of the revision, the selection of the text to revise or the positioning of the cursor, the (possible) deletion of the text and the end of the revision. In Figure 10 we describe two (technical) revision operations to change the last word of the sentence 'Questions of science, science and progress.' into 'evolution'.

Questions of science, science and [progress.]$^{1}$|$^{1}$ {evolution.}

The first operation is a very basic one: the writer simply uses the backspace key at the point of utterance to delete the full stop and the word 'progression' and then types the new word 'evolution' (see rule 1). This is a rather minimal operation, because the writer does not have to move nor position the cursor in the text produced so far. However, the writer could also opt for another sequence to realize this substitution: he can move the mouse to the left, position the cursor by clicking on the left

[7] For more information about Morae we refer to the Techsmith's website: www.techsmith.com.

[8] We would like to thank Stijn van Even, Guido Gallopyn and Neil Grant of Scansoft for all their efforts in making the logging facility at Dragon Naturally Speaking available to us.

button on the mouse, use the delete key to delete the word, change the text and move to the point of utterance by using arrow keys to the right (see rule 2).

| | begin (movement) | selection/positioning | deletion | end (movement) |
|---|---|---|---|---|
| 1 | - | - | backspace | - |
| 2 | mouse movement left | left click | delete | arrow right |
| n | … | … | … | … |

Figure 10: Example of rules to define revisions

At the moment we have predefined about 50 sets of rules to test the algorithm for deletions and substitutions. However, after the testing phase, the rules will have to be extended and further tested to cover a more complete range of revisions.

## 7.3. Progression analyses

At this moment the development of the text is represented in the linear text analysis. To visualize this textual development we would like to extend Inputlog with two graphical representations of the text progression, a basic and a more extended one. In the basic progression analysis we would like to visually represent the number of characters that are produced at each moment during the writing process taking into account the characters that are deleted at that stage. This basic progression analysis is based on writing strategies research by Perrin (2003).

Because this basic analysis is a static reproduction of the writing process, we would also like to develop a more interactive representation inspired by Lindgren (2005). She uses a Geographical Information System (GIS) to visualize and summarize the writing process. GIS enables researchers to analyze different subprocesses of the writing process by selecting representative variables. The graphical representations are not static, but they allow a researcher to interact with the data at different levels and to move back and forward between the data and their representation. Consequently, as well as being a tool for visualization and data-mining, this technique can support a dynamic analysis of the cognitive processes during writing due to the interactive nature of the data-mining approach on which GIS is based. Figure 11 shows an example of a GIS graph that is based on a manually adapted dataset of Inputlog[9]. To generate these kind of progression analyses automatically, we first need to further optimize the revision analysis because these are based on the revision output.

On the x-axes the time (in sec) is represented; on the y-axes the number of characters that are produced cq. realized effectively in the text produced so far are indicated. The top line indicates the total character production including deleted characters at each point in time; the bottom line indicates the characters retained after deletions at each point in time. The dotted line shows all the points in time at which the writer is working on the text, representing both pauses and deletions. The size of the circles refers to the length of the pause. When the line drops, a number of characters are deleted.

[9] We would like to thank Eva Lindgren of the Department of Modern Languages, Umeå University (Sweden) who generated this graph for us with ArcGIS (ESRI).

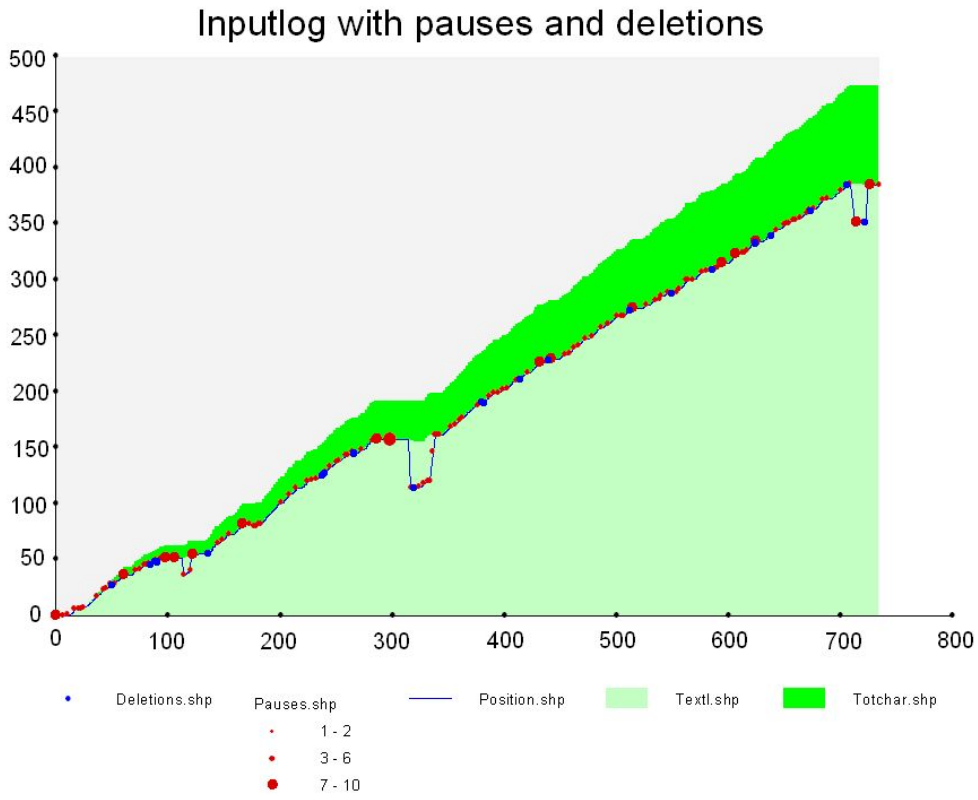



Figure 11: graphic representation of text progression logged in Inputlog and visualised in a GIS application (source: Lindgren)

### 7.4. Integration with Morae

Inputlog is designed for micro analytic research on writing processes. However, these very detailed data can easily be combined with more macro analytic research tools. Therefore, we would like to complement the data of Inputlog with another observation tool, for example, Morae. This program is mainly developed for usability testing and uses an online screencam (Camtasia) to register every action on the screen. However, next to some lower level analyses, Morae also captures changes between programs on a higher level and registers, for instance, the url-addresses of websites that are accessed during a writing session. Just like Inputlog it also logs very detailed timestamps which should enable us to integrate the additional data registered by Morae into the output of Inputlog. For the observation of writing processes during which the participants combine Word with other programs especially, this integration opens new perspectives for further analyses.

## 8. Conclusion

In this paper we have shortly described the main characteristics and the functionality of Inputlog. The program differs from other keystroke logging programs in a way that it is not limited in its usage to a self designed word processor. It is primarily developed to log writing processes in Word (Windows environment).

Inputlog offers three main functions - record, generate and play – enabling the researchers to collect very detailed data about a writing process and to prepare some basic analyses for further study. The replay function allows for a review of the writing process and it can be also be used as a stimulus for a

retrospective thinking aloud protocol. For more detailed information about the use of the program we refer to the help file and the detailed desciption on the program's website. [10]

To increase the applicability of Inputlog we would like to further develop the program by adding new modules. Four new components are planned: a module to log speech recognition events, a module for revision analysis, a module for progression analyses and a module to integrate macro-level data recorded by Morae. We hope to report about these further developments soon.

## Acknowledgement

We would like to thank Wesley Cabus, Ahmed Essahli and Bart Van de Velde for their excellent work in programming Inputlog. Thanks also to Tom Van Hout for editing and proofreading an earlier version of this paper.

[10] See: http://webhost.ua.ac.be/mleijten/inputlog

## References


Eklundh, K.S. (1994). Linear and Non-linear strategies in computer-based writing. *Computers and Composition*, *11,* 203-216.

Eklundh, K.S. & Kollberg, P. (1992). *Translating keystroke records into a general notation for the writing process*. Royal Institute of Technology, Stockholm: IPLab, Department of Numerical Analysis and Computing Science.

Kollberg, P. (1998). *S-Notation - a computer based method for studying and representing text composition.* Stockholm: Stockholm University.

Leijten, M. & Waes, Van L. (2003). *The writing processes and learning strategies of initial users of speech recognition: A case study on the adaptation process of two professional writers.* Antwerp: Faculty of Applied Economics, University of Antwerp (Research paper).

Leijten, M. & Van Waes, L. (2005, accepted for publication). Writing with Speech Recognition: The Adaptation Process of Professional Writers. *Interacting with Computers*, 17/5.

Leijten, M., Ransdell, S. & Van Waes, L. (in preparation). Working Memory and Error Detection in Writing: A task analysis of error size, lexicality, and mode of writing.

Lindgren, E. (2005, to be published). GIS for writing: applying geographic information system techniques to data-mine writing's cognitive processes. In D. Galbraith, M. Torrance and L. Van Waes, *Writing and Cognition.* Amsterdam/Boston: Elsevier.

Perrin, D., (2003). Progression analysis: Investigating writing strategies at the workplace. *Journal of Pragmatics, 35*, 35 (6), 907-921.

Spelman Miller and Sullivan (2006, to be published). Keystroke logging: An introduction. In K. Sullivan & E. Lindgren, *Computer Key-Stroke Logging.* Amsterdam/Boston: Elsevier.

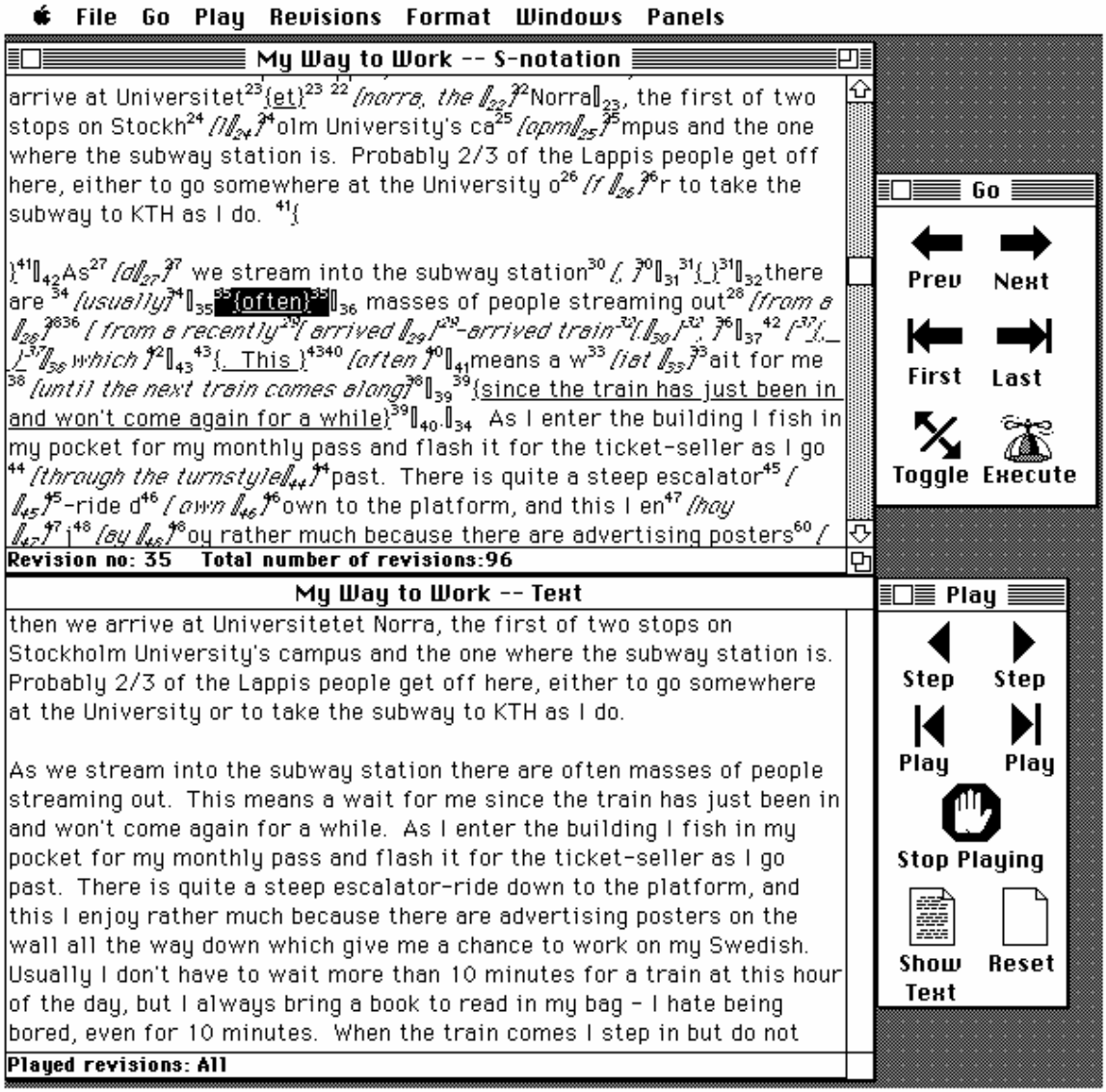
Severinson Eklundh, K & Kollberg, P. (1996). Computer tools for tracing the writing process: From keystroke records to S-notation. In G. Rijlaarsdam, H. van den Bergh & M. Couzijn, *Theories, models and methodology in writing research*, pp.526-541. Amsterdam: Amsterdam University Press.

Severinson Eklundh, K. & Kollberg, P. (2003). Emerging discourse structure: Computer-assisted episode analysis as a window to global revision in university students' writing. *Journal of Pragmatics, 35* (4), 869-891.

Strömqvist, S. & Malmsten, L. (1997) *ScriptLog Pro User's manua*l. Göteborg University, Dept of Linguistics.

Sullivan, K. & Lindgren, E., Eds. (2006). *Computer Key-Stroke Logging.* Amsterdam/Boston: Elsevier.

# Appendix 1: Interface of Trace-It



# Appendix 2: Glossary

| | |
|---|---|
| **scancodes** | The data from a keyboard comes mainly in the form of scancodes, produced by key presses or used in the protocol with the computer. The PC keyboard interface is designed so the system software has maximum flexibility in defining certain keyboard operations. This is accomplished by having the keyboard return scancodes rather than ASCII codes. Each key generates a 'make' scancode when pressed and a 'break' scancode when released. The computer system interprets the scancodes to determine what operation it is to perform. (http://www.barcodeman.com/altek/mule/scandoc.php) |
| **journalrecord** | public static final Hook.Descriptor JOURNALRECORD<br>Records input messages posted to the system message queue.<br>(http://www.jniwrapper.com/docs/javadoc/winpack/com/jniwrapper/win32/hook/Hook.html) |
| **journalplayback** | public static final Hook.Descriptor JOURNALPLAYBACK<br>Posts messages previously recorded by a JOURNALRECORD hook procedure. |
| **event** | action on the computer |
| **EVENTMSG** | The EVENTMSG structure contains information about a hardware message sent to the system message queue. (This structure is used to store message information for the JournalPlaybackProc callback function.) |
| **Windows Hook** | A hook is a point in the system message-handling mechanism where an application can install a subroutine to monitor the message traffic in the system and process certain types of messages before they reach the target window procedure. |
| **GUI** | Graphical User Interface |
| **CPU** | Central Processing Unit<br>It reads instructions from your software and tells your computer what to do. |
| **DLL** | Dynamic Link Library<br>a Windows library that can be shared by multiple applications |
| **ArrayList** | Implements the **IList** interface using an array whose size is dynamically increased as required. |
| **Virtual Keycode** | unique number that identifies one key |

# Appendix 3: Analyses

| analyses | output | explanation |
|---|---|---|
| **general logging file** | Microsoft Excel spreadsheet<br>(full description and example in appendix) | basic log file of the writing session |
| **statistical analysis** | Microsoft Excel spreadsheet<br>(full description and example in appendix) | basic statistical information on the writing session |
| **text analysis** | Text file with the final text at the end of a writing session (without lay-out, font, etc.)<br>This file is generated automatically | final text<br>(example in appendix) |
| **linear text** | Text file with the complete linear production of the text including mouse movements and other activities | linear text<br>(full description and example in appendix) |
| **linear text + modes** | Same text file as the linear text, but now the different writing modes are included | linear text & writing modes |
| **linear text (periods)** | Text file with the complete linear production of the text including mouse movements and other activities of the writing process divided into periods (free to choose) | linear text in periods<br>(example in appendix) |
| **linear text (intervals)** | Text file with the complete linear production of the text including mouse movements and other activities of the writing process divided into intervals (free to choose) | linear text in intervals<br>(example in appendix) |
| **pause analysis** | Microsoft Excel spreadsheet with<br>analyses of every non-scribal period of 1 second and longer | 1 second |
| | Microsoft Excel spreadsheet with<br>analyses of every non-scribal period of 2 seconds and longer | 2 seconds |
| | Microsoft Excel spreadsheet with<br>analyses of every non-scribal period of 5 seconds and longer | 5 seconds |
| | Microsoft Excel spreadsheet with analyses of every non-scribal period of any – user-specified - pause length | number of seconds free to choose |
| **mode analysis*** | Microsoft Excel spreadsheet | basic writing mode information on the writing session |
| **revision analysis*** | Microsoft Excel spreadsheet | information on different levels about the revisions in the writing session |

* not active in Inputlog version 1.4

# Appendix 4: General logging file

| column | labels | variables |
|---|---|---|
| **a** | **number** | set value: 1, 2, 3, 4, …. 998 etc. Every row is defined by a unique number |
| **b** | **participant** | number assigned to the participant |
| **c** | **sex** | sex of the participant (e.g. 0=man, 1=female) |
| **d** | **session** | number of the experimental session (e.g. 1= task 1, mode 1, 2 = task 1, mode 2…) |
| **e** | **group** | type of profession (e.g. 1=academic, 2=lawyer) |
| **f** | **experience** | experience with writing modes (e.g. 1=speech, 2=dictating, 3=keyboard&mouse) |
| **g** | **age** | age of the participant |
| **h** | **…** | Optional user defined variables: maximum 4 |
| **i** | **writing mode** | input mode (1=keyboard, 2=mouse, 3=speech) |
| **j** | **output** | text that appears on the screen |
| **k** | **input in** | time of key in: hours:minutes.seconds,100$^{th}$ of a second |
| **l** | **input in_sec** | time of key in: in seconds |
| **m** | **input out** | time of key up: hours:minutes.seconds,100$^{th}$ of a second |
| **n** | **input out_sec** | time of key up: in seconds |
| **o** | **action time** | time between key down and key up: hours:minutes.seconds,100$^{th}$ of a second |
| **p** | **action time_sec** | time between key down and key up: in seconds |
| **q** | **pause time** | time between key down and key in: hours:minutes.seconds,100$^{th}$ of a second |
| **r** | **pause time_sec** | time between key down and key in: in seconds |
| **s** | **x value** | location of mouse on x-axis |
| **t** | **y value** | location of mouse on y-axis |

**!** Only the last two columns contain the values of the mouse. The location of the keystrokes are not detailed.

# Appendix 5: Linear text

The output of the linear text production consists of 2 elements:

- output (characters as they appear on the screen).
- function keys (they are indicated by square brackets [ & ]).

In the output the following keys are shown:

| **Action** | **Label in [ ]** |
|---|---|
| Interspace (visualized by output) | **SPACE** |
| Backspace (+ number of characters or keystrokes) | **BS** |
| Delete (+number of characters) | **DEL** |
| Shift (on/off; visualized by OUTPUT) | |
| CapsLock (on/off; visualized by OUTPUT) | |
| Control (+ key) | **CTRL** |
| Alt (+ key) | **ALT** |
| Up (+ number of lines) | **UP 6** |
| Down (+ number of lines) | **DOWN 8** |
| Left (+number of characters) | **LEFT 15** |
| Right (+ number of characters) | **RIGHT 1** |
| Tab (+ number) | **TAB** |
| Enter (+ number) | **ENTER** |
| Home | **HOME** |
| End | **END** |
| PageUp (+ number) | **PU** |
| PageDown (+ number) | **PD** |
| Insert (+ characters) | **INS \| INS STOP** |
| Mousemovement without click (+ start and end value of xy-axis+segment) | **MwC.56\|1-78\|6** |
| Mouseclick left | **McLeft** |
| Mouseselection (click-movement-unclick & double-click) | **Mselect** |
| Scroll (with wheel & bar) (+start and end value of xy-axis+segment) | **Mscroll** |
| Mouseclick right | **McRight** |