Precise Timing Data for the Use of a Single Button

Gautam Bhatnagar

Submitted in part fulfilment of the BSc in Computer Science.

Department of Computer Science

THE UNIVERSITY of York

March 2003

Supervised by Alistair Edwards

Number of words = 7913 as counted by Microsoft Word's word count function
This includes the body of the report, but excludes the appendixes

1. Table of Contents

2. Table of Figures

3. Acknowledgements

I would like to thank my supervisor, Alistair Edwards, for his guidance and Richard Pack for all of his help with the hardware. Additionally I would like to thank all my housemates for their patience, support and proof reading skills.

4. Abstract

Modern portable devices, such as mobile phones, watches etc. rely on the input that can be achieved using a small number of buttons. To maximise the range of inputs it is important to use these buttons in as many ways as possible, e.g. using double clicks or triple clicks etc. to provide new meanings. When developing consumer devices it is necessary to know the precise timing parameters to be applied to these different types of button presses. It was the objective of this study to obtain precise timings for these parameters. Data was collected from average to above-average users within the age group 15 to 50 year olds and these results are presented. The results can be used as a basis for interaction designs or for further research. One of the conclusions drawn is that there is a wide range of button-pressing performance in the population and that systems which adapt to their individual users may be more useful although values for each click are presented, with the error rates that those values might incur.

5. Introduction

Traditionally and historically, buttons have been used as a way for people to interact with electronic systems and other machinery. There are many portable devices in use today (e.g. watches, mobile phones, portable music players etc.); in such applications, miniaturisation is a much prized commodity. There is, however, a limit to how small these devices can be made. One limitation of this miniaturisation process is the number of buttons (or other input devices) that must be present for adequate interaction with the device.

Despite still being one of the primary methods of communicating with devices, there has been little research published recently which has paid attention to the use of buttons. While, previously it could be said that *"interfaces are an unappreciated piece of the technology puzzle"* (David Levy - former head of Apple's ergonomic design division [BBC News, 2002a]) there is an increasing focus on interface design to increase productivity and usability, but much of the research remains out of the public domain. This is possibly because much of the work is done for the development of commercial products, and as such the companies involved do not feel compelled to make their results available to their competitors.

The demand for this research comes from users of small portable devices increasingly wanting (or needing) to input many and complex messages to their devices.

For example, mobile phones commonly have to use a ten or twelve button keypad to input fifty-two upper and lowercase letters, in addition to spaces, punctuation and controls. Therefore it makes sense to "make the most" of the number of buttons available by trying to use them in a variety of ways.

Figure 5.i Fastap Keypad (picture courtesy of BBC News, 25th November 2002)

There has been research performed by persons such as David Levy (the former head of Apple's ergonomic design division) to find a way to fit individual buttons for as many letters as possible. Levy's method, the Fastap keyboard (**Figure 5.i**) works by giving each character each own button, with numerals being produced by pressing the surrounding four buttons [BBC News, 2002b]. This is particularly useful for reducing the number of presses using "traditional" text messaging systems (such as predictive text software), especially for languages such as Japanese where current systems take up to 8 presses to produce some of the 120 characters in the Japanese alphabet [BBC News, 2002a].

Despite these innovations there is a limit to how many buttons can be fitted on a small device due to ergonomic factors. Therefore there needs to be some way of multiplying the use of a single button.

"There are variations of elementary pressing of the button (double-clicking etc) which can be used to broaden the input, but there is a question as to how many different button presses are practical." [Edwards & Li, 2000]

There are many theories available, which are useful in predicting and explaining human performance, errors, ease of learning etc. The Model Human Processor [Card et al., 1983] provides one way of predicting human performance in human-machine interactions and can give us guidelines as to what the timing might be. However, a theory is just that; and as such, obtaining the timing data required, in the accuracy required can only be done by an empirical human experiment.

Yanyu Li [Li, 2000] attempted to answer the question of how many presses are practical by providing detailed results, from actual user inputs, of the time taken to perform a selection of different ways of pressing a single button (a single click, a double click, a triple click and a long press in the case of her experiment) in an attempt to answer this question.

However Li's project suffered from major inaccuracies in the method used to record the timings, as the experiment was performed using a software only solution which had an accuracy of 50ms (in some cases this would make the possible error over 100%).

This project is based heavily on Li's but is mainly concerned with collecting timing data with increased precision, for the different types of button presses in order to provide guidelines for appropriate time parameters for real world usage. Our project, however, used a hardware

based timing mechanism which increased the accuracy considerably. The hardware device was built in the department and skeleton code was constructed in Visual C++ to interface with the device. Part of this project was to construct a graphical user interface to communicate with the device, and part was to perform experiments and obtain user data using the graphical interface.

There is an importance to strike the right balance for these timings. Slow timings extend the time of the interaction, and can frustrate the user. Faster times may, however, result in errors (due to faster reaction times required for the actions) and once again increase the time of the interactions (due to repetitions) and frustrate the user. This project aims to produce a range of optimum timings which would give an appropriate balance between performance and error rates for a generic single button situation.

6. Previous Work

Much of the research on previous work relating to this project has been performed by Li [Li, 2000] and this work is summarised here in this chapter.

Yanyu Li defined a button as being *"a tool, a bridge that connects the user and the machine, in order to realize the communication between them"* [Li, 2000]. There are countless buttons present in the modern world, all of varying sizes, shapes and even textures; there are even simulated (graphical) buttons present in computer software.

The uses of these buttons are sometimes not as simple as a single press, and sometimes involve multiple presses, or possibly a combination of other buttons being pressed as well, to get the desired action. Additionally buttons may have multiple meanings, which change during the course of the user's interaction - these are sometimes known as "soft" keys.

Of course, interactions with devices are not limited to buttons (in their myriad forms); one of the commonest ways to interact with computers is a mouse, which consists of a method (optical, mechanical or otherwise) of translating the mouse's movement into some meaningful action for the computer (e.g. movement of a cursor). The mouse usually has a number of buttons present as well.

6.1. Multiple buttons versus a Single button

Although Gregg Williams' tests using Apple Computer's Lisa indicated *"that people aren't always sure which button to push on a multiple-button mouse"* [Williams, 1983], a project investigating the error rates and performance of users when using multiple buttons was performed by Lynn Price [Price, 1983]. In this project users were asked to indicate true or false with either a single click or double click in the first case, and with two different buttons in the second case; the conclusions drawn were that people performed faster and with lesser errors when using separate buttons for each action (indicating true or false in this case).

However it may be impractical to fit many buttons on small devices; the users of these devices still expect to be able to communicate complex messages with their devices. Therefore there needs to be a compromise; despite the conclusions of Price [Price, 1983] pointing to multiple buttons being more efficient, multiple ways to use a single button need to be found.

One method of providing multiple uses of a single button is that of “soft” keys, where the meaning of the button press changes as the user proceeds in their interaction. The meaning is sometimes indicated on a display (if one is available). This system is commonly used on mobile phones and PDA devices. There are other methods involving pressing different combinations of buttons to obtain different results but, as anyone who plays videogames will tell you, these combinations can be tricky to master and are not error free.

One of the simplest methods of increasing functionality of the buttons is that of multiple presses, it is that which was investigated by Li [Li, 2000] and will be investigated by ourselves.

6.2. Uses of a Single Button

Yanyu Li [Li, 2000] investigated the following types of different button presses:

- Single-click
- Double-click
- Triple-Click
- Long-Click (Extended Click)

6.2.1. Single-Click

This is the commonest way of using a button; the button is pressed then released at once. This is shown in the **Figure 6.i**, where one bit is transmitted, the only parameter being c, which is the length of time that the button had been depressed for.

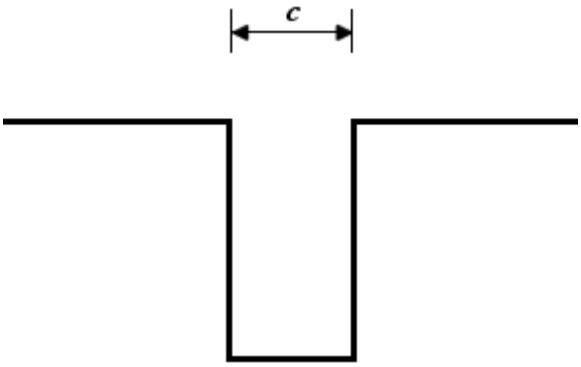


Figure 6.i A Single Click

6.2.2. Long-Click (Extended Click)

This is a sustained click i.e. the button is depressed, held in that position for some time and then released. This differs from the single click in the duration that the button has been depressed for, ***l***.

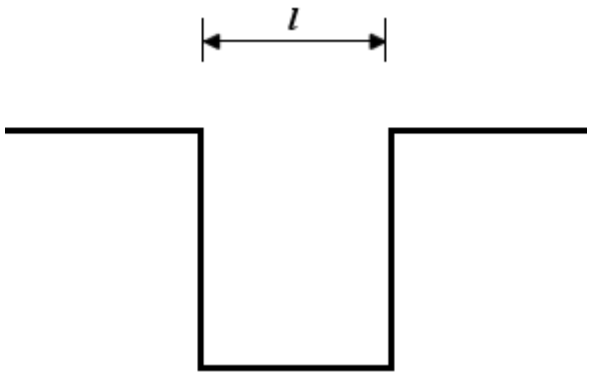


Figure 6.ii A Long Click

6.2.3. Double-Click

This is two rapid clicks but differs from two single-clicks in meaning and in timing. We will try and quantify the three timing parameters ***d1***, ***d2*** and ***d3***, so as to form a characterisation of a double-click. The parameter ***d2*** is crucial to decide what is determined to be a double click and what is to be taken as two single-clicks [Li, 2000].

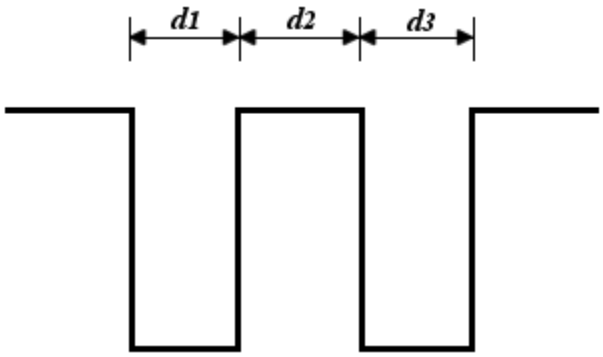


Figure 6.iii A Double Click

6.2.4. Triple-Click

This is an extension of the double-click. In this case there are five timing parameters, ***t1***, ***t2***, ***t3***, ***t4***, ***t5***. The parameters ***t2*** and ***t3*** being crucial to determine whether it is a triple click, three single clicks or a combination of a double and single click [Li, 2000].

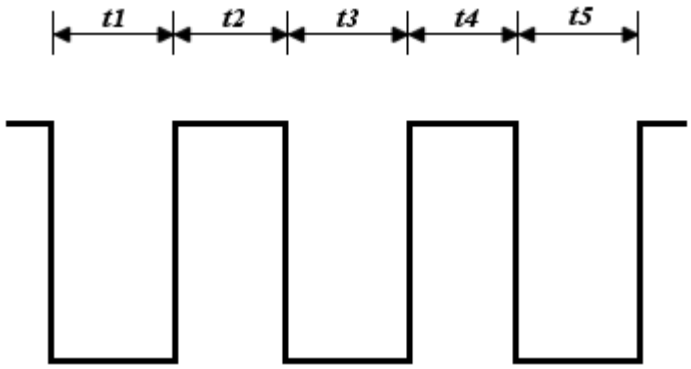


Figure 6.iv A Triple Click

6.3. Human-Machine Interaction Psychology

6.3.1. The Model Human Processor

The Model Human Processor Theory [Card et al., 1983] describes and attempts to explain human performance. This model views the human mind as an information-processing system which is built up of a set of models to describe and predict human reaction times in different operations. The interacting processors modelled are the perceptual processor, the motor processor and the cognitive processor, all linked to memory. The innards of these processors are not what we are concerned with however, and so the most important parameter of each of these processors is T, the cycle time i.e. the time taken for each processor to perform their action. Given that human brains (and human beings!) are very variable objects and there are many uncertainties to be dealt with, three separate models have been created.

These are *Slowman* (worst performance), *Middleman* (normal performance) and *Fastman* (best performance). The timings for each of the processors cycle time is presented in **Table 6.i.**

	Slowman	Middleman	Fastman
Perceptual Processor Cycle (T_p)	200ms	100ms	50ms
Cognitive Processor Cycle (T_c)	170ms	70ms	25ms
Motor Processor Cycle (T_m)	100ms	70ms	30ms

Table 6.i Processor cycle times adapted from "The Psychology of Human-Computer Interaction", Card et al. 1983

These timings provide us with a way of predicting the performance in our button presses.

6.3.2. Timing Predictions

This still leaves us with two ways of modelling our button presses, depending on the level (if any) of cognition required for the button presses. The level of cognition required could depend not only on the complexity of the button press (although this should not be an issue as we are only investigating relatively simple button presses) but also on the user's familiarity with those button presses.

This distinction affects the predictions significantly as detailed below.

Assuming the action is purely dependent on the Motor System (processor) for a single button press, then the time is $T_m + T_m$ [Card et al., 1983] (time to press button, time to release button) i.e. the stroke (c) will last 140ms (60ms for *Fastman*, 200ms for *Slowman*).

Alternatively if we need to incorporate the perceptual and cognitive cycles then the time take will be $T_m + T_p + T_c + T_m$ = 310 ms {135 – 470}

The double-click and triple-click predictions should merely be an extension of this. However the long-click would certainly involve the perceptual and cognitive cycles due to the user having to determine when the time button had been pressed long enough (in real world situations the user may receive some kind of feedback, e.g. auditory or visual, that the action is complete and then release)

6.4. Summary and Analysis of Results from Yanyu Li's Experiment

The results from Yanyu Li's Experiment are presented below. The groups were segmented on age. In general similar readings were taken for the age group of 10 to 50 years, but with noticeably slower times for those outside the age range. The results are discussed in further detail below.

Li also made a cursory investigation into error rates for each of the clicks, it is unclear from her report how these errors were trapped, but the errors were listed as:

- making a long click as a single click or vice versa
- making a double click as a single click
- making a triple click as a double click
- making a double click as a triple click

The error rates for each click are given in the sections following. Jonathan Tuffin [Tuffin, 2001] carried out a more detailed investigation into error rates, where he used values from Li's experiment to produce a time limit for each click based on a 40% error rate. The users were surveyed as to their state of mind after the test, and were found to be quite resilient and accepting of errors.

6.4.1. Single-Click

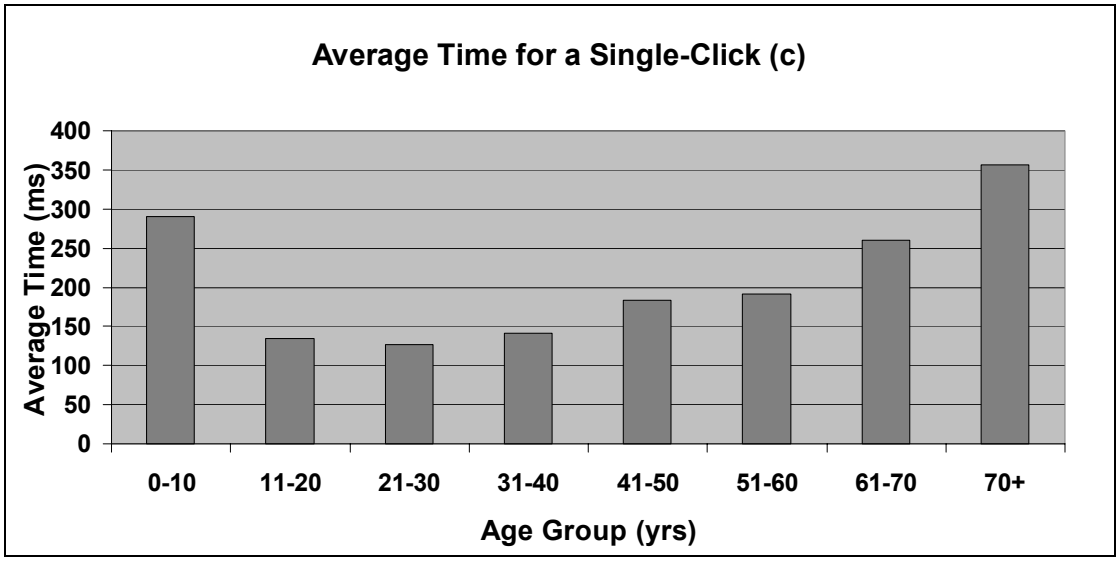


Figure 6.v Results for c from Li's experiment

The mean across the age groups was calculated as 192 ms. For the age group of 10 to 50 years the average of approximately 140 ms seems to bear out the initial prediction (with no perceptual cycle involved) of 120 ms from the Model Human Processor. Additionally results from the groups in this age region were closer to the normal distribution.

Error rates for this click were also recorded with a rate of 1.3% of the total single clicks made by 12.1% of the users.

6.4.2. Long-Click

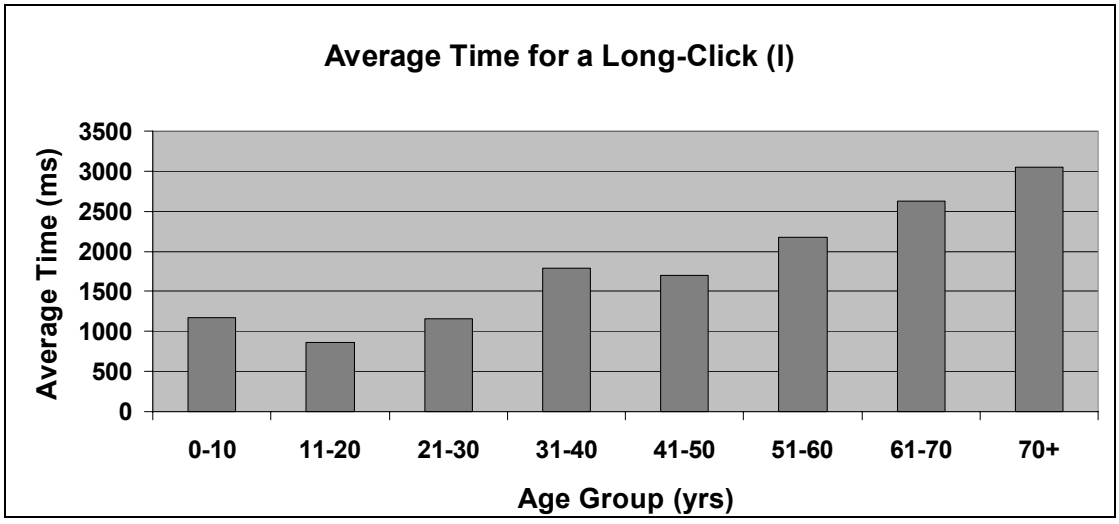


Figure 6.vi Results for l from Li's experiment

The mean across the age groups for ***l*** was 1684 ms with a mode of 880 ms, a median of 1160 ms and a standard deviation of 1863 ms. The long click had an error rate of 2.8% made by 25% of the users.

6.4.3. Double-Click

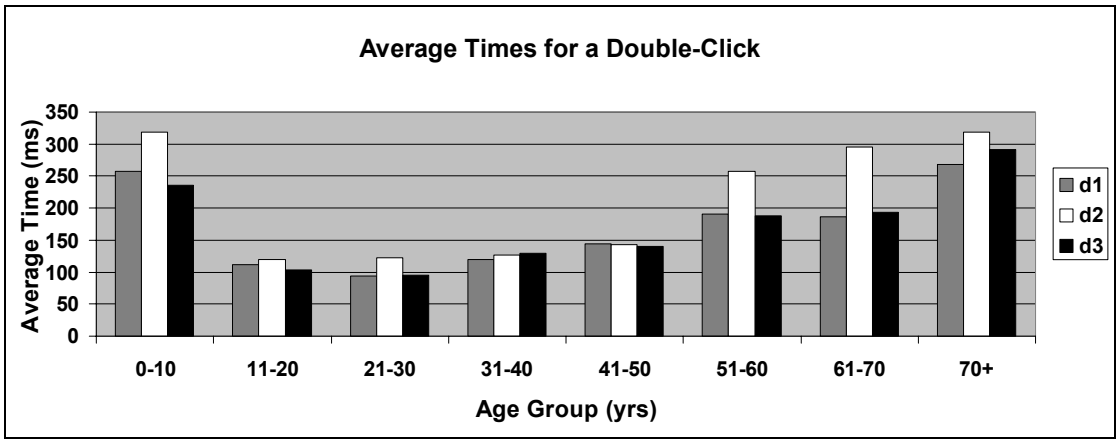


Figure 6.vii Results for d1, d2 and d3 from Li's experiment

The average time for ***d1*** was 160 ms; ***d2*** had an average of 196 ms and ***d3*** had an average time of 158 ms, with an average time of 514 ms for the total click length. Again the age group of 20 to 40 year olds showed a closer match to the normal distribution (with the mode, median and mean closer to each other).

The error rates observed were 3.2% made by 25% of users.

6.4.4. Triple-Click

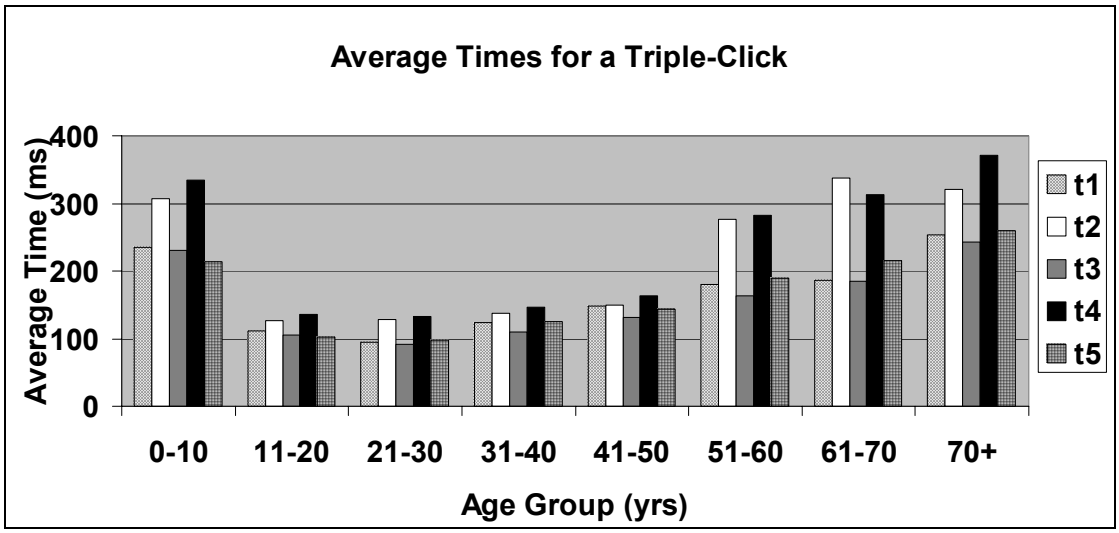


Figure 6.viii Results for t1, t2, t3, t4 and t5 from Li's experiment

The observed averages were: 156 ms for ***t1***, 205 ms for ***t2***, 147ms for ***t3***, 217 ms for ***t4***, and 156 ms for ***t5***. Again, the middle age group of 10 to 40 years old produced closer to normal

distributions, with modes, medians and means of approximately 110 ms. The error rates were 4.4% made by 30% of the users.

6.4.5. General Conclusions from Li's Experiment

The averages of the results seem to bear out the *Middleman* predictions, however more directly from the data we can see that there were a number of people with timings more in line with *Fastman* predictions (80 ms for c, $d1$, $d3$, $t1$, $t3$, $t5$). However the results for slower timings were out of the *Slowman* range predictions; this could be due to the results from those users being affected by unfamiliarity with the task, or lack of clarity or understanding in what they were supposed to do.

6.5. Problems with Li's Experiment

The major problem with the data collected so far by Li, is the accuracy of the results. Li's project used software to collect the data which was only accurate to 50ms. This is particularly unacceptable due to the duration of the clicks. In some cases the inherent error from using software would be over 100%! We will attempt to ameliorate this problem by using a hardware timer to record the timing information and communicate with a computer which will collect and collate the data.

This was not the only problem with the experiment. Additionally there seemed to be some ambiguity in the experimental instructions (especially some confusion over the definition or meaning of the long-click). We will also attempt to remedy this with clearer instructions.

The errors which users made were also not automatically trapped we will attempt to remedy this by providing some form of error catching in software.

The last significant modification we would like to make to the experiment is to the age range that it will be performed upon; Li's experiment covered a very varied age range and the conclusions reached from the results were that the timings were in general homogenous, apart for slower timings in persons aged under 10 and over 60 [Edwards, 2002]. Our experiment will concentrate on a intermediate age group of 15 to 50 year olds.

7. Experimental Design

7.1. Hardware Design

7.1.1. Switch Device

A hardware device was constructed using a PIC16745 controller (available from Microchip (further details, such as datasheets etc. are available at http://www.microchip.com/, the schematic is presented in Appendix D). This was attached to a switch and was used to record the timings for that switch. The de-bouncing of the switch and timing collection is all done in the PIC software. It interfaced with the computer over a Universal Serial Bus (USB) connection. The switch itself was a micro-switch, with a small plastic cap (**Figure 7.i**).

Figure 7.i Black switch used in the experiment

The device stored the time take between any depression or release of the switch i.e. for a single click, the device would first record data for the time between the start of the single press and the last click (this data is not pertinent to our experiment) and the time between the release of the button and the start of the button press – this would be the value c that we are looking for.

This works similarly for each of the other types of click we want to record, with the first value being the time between each different type of click. i.e. for a long click the first timing returned would be irrelevant for our experiment, the subsequent timing being l; for a double click, the next timings retuned would correspond to $d1$, $d2$, $d3$; for a triple click the subsequent timings would be $t1$, $t2$, $t3$, $t4$, $t5$.

7.1.2. Computer

The computer used was a Sony Vaio PictureBook C1-MHP. It was sufficiently specified so as to run the test quickly and with minimal delay. The switch device was connected to the computer via the Universal Serial Bus ports present on the machine. The device communicated via this connection and was powered by it. Using a laptop was necessary as it is easier to go to the users, rather than to try and get them to come to you.

7.2. Software Design

7.2.1. Original Software Code

"Skeleton" software was coded by Richard Pack in Visual C++, modified from Jan Axelson's book "USB Complete: Everything You Need to Develop Custom USB Peripherals" [Axelson, 2000]. This provided the routines for detecting the device and retrieving the timing data from it. This code was modified to provide a graphical user interface for performing the test. Because the initial code was provided in Visual C++, it was decided to maintain that language to implement the test software rather than translate the code into a different language.

There was, however, a choice to be made of which operating system the software was to be run of. Initial construction and testing of the skeleton code was done on Microsoft Windows 2000. When the code was migrated to Microsoft Windows XP (the platform of the Sony laptop) it was found to crash.

The cause of this was narrowed down to a possible Human Interface Device (HID) driver issue or, more likely, caused by a difference in the way Windows XP handles the Windows Message Timer (WM_TIMER) command (in Windows XP this is a low priority message and because XP has many more background processes running, this causes the program to not respond in time).

This problem could have been overcome by writing a separate thread to perform timer actions to replace the WM_TIMER but this method was eschewed for simplicity and the decision was taken to migrate to Microsoft Windows 2000 instead as there was no guarantee that further problems (such as the possible driver issue) would not manifest themselves.

7.2.2. The Test

The software was coded with the purpose of conducting the experiment in mind. As such, some of the niceties provided by the Microsoft Foundation Class (MFC) libraries (file input / output, menu items etc.) have been dropped so as to reduce any complexity for the test subject / user and "guide" them toward completing the test. The coding itself was very straight forward with no major design decisions required. All the data collected was of fixed lengths and so was then implemented in arrays for simplicity, speed and ease of access. The code is presented in Appendix E for convenience.

The user was presented with a set of instructions to read through before performing the test; these instructions described the test and the actions the user was to perform (single click, double click, etc). These instructions are in Appendix C.

The following images are examples of the screens presented to the user during the test.

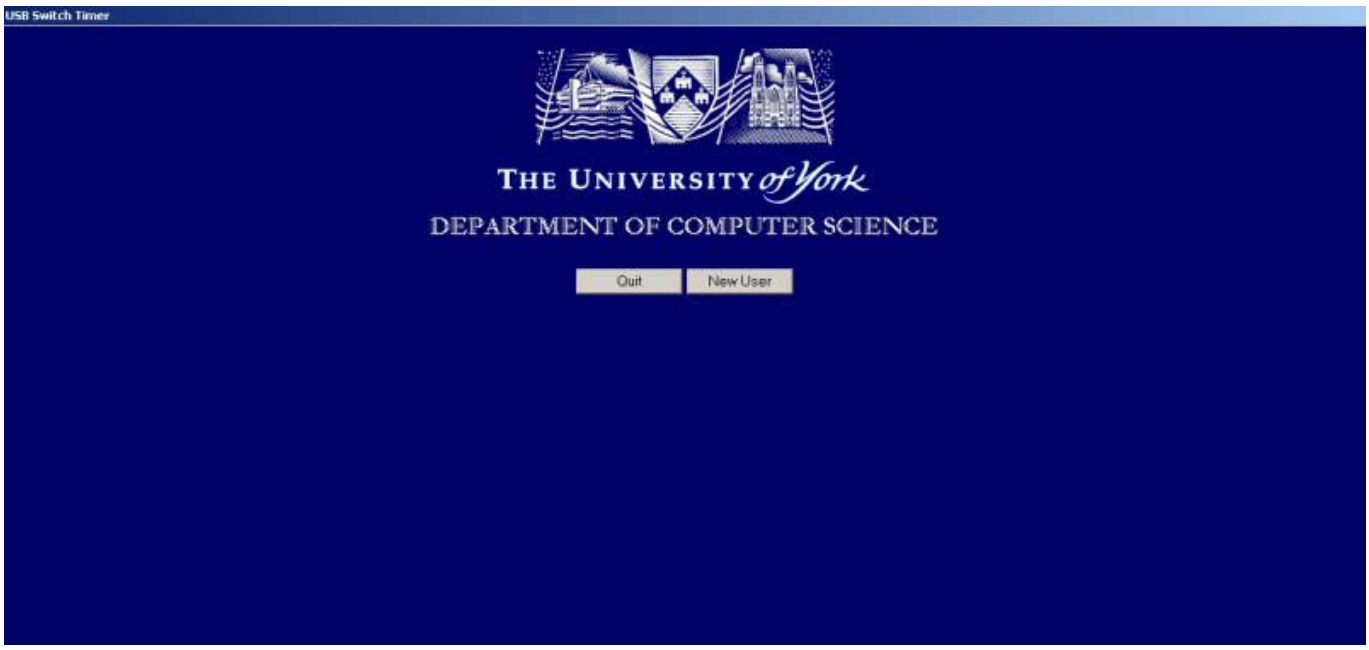


Figure 7.ii The first test screen presented to the user

The user is first presented with an opening screen from which they can select to start a new test or, if they wish, to quit (**Figure 7.ii**).

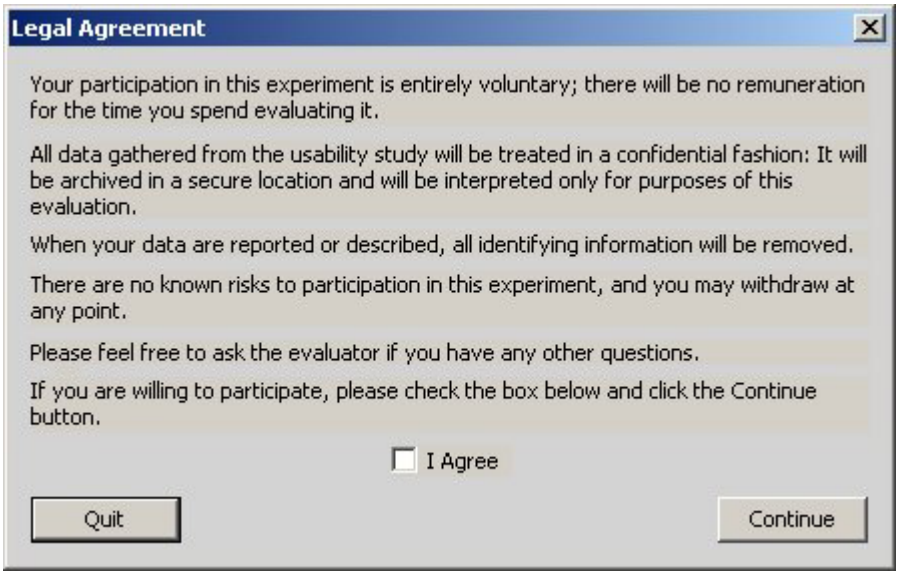


Figure 7.iii The legal statement to which the user must agree

Starting a test brings up a dialog box, which contains a legal statement they must agree to continue with the test (**Figure 7.iii**).

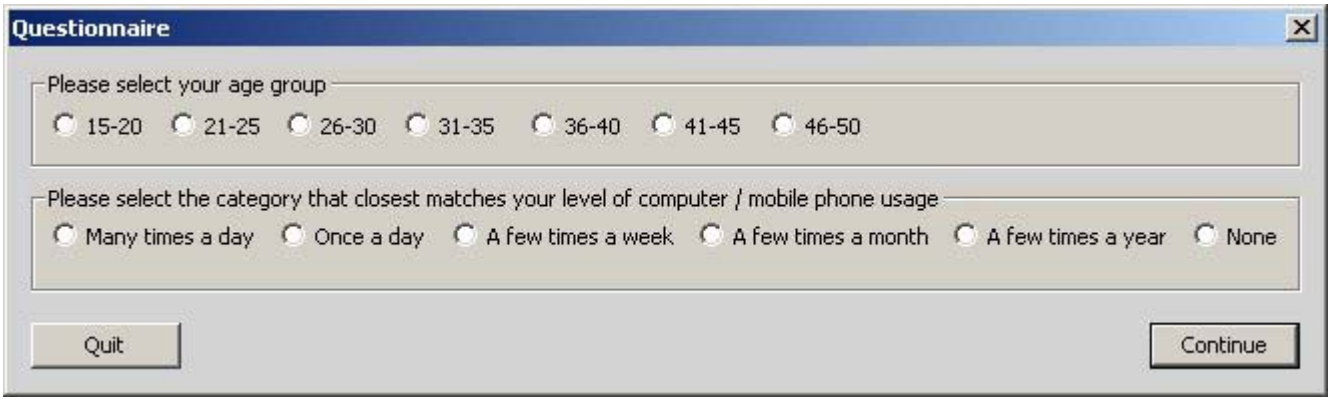


Figure 7.iv The questionnaire the user must complete to continue with the test

Having agreed to this the software then asks for statistical information about the user, that being their age and the level of computer / mobile phone usage (**Figure 7.iv**). These categories were chosen due to the being the most pertinent characteristic for user segmentation in Li's experiment (although the age range was chosen specifically to lessen the variation in the timings obtained).

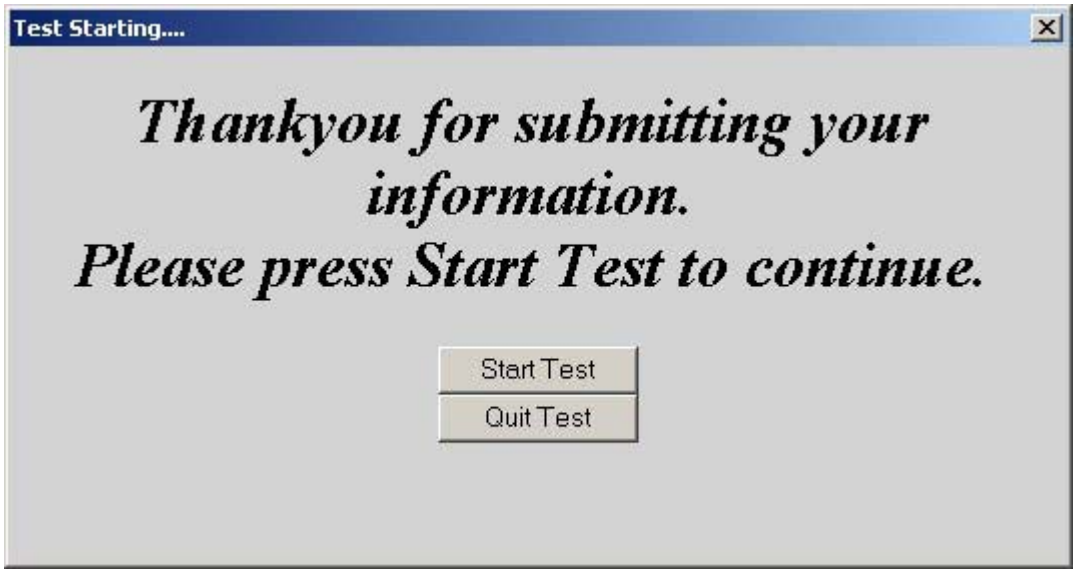


Figure 7.v The test from which the user will launch the test

Once the user has selected their information they can proceed to the next screen where they can start the test (**Figure 7.v**).

Figure 7.vi A sample of the prompts for clicks, in this case a triple click

The test consists of the user performing each of the four types of click (detailed earlier) ten times. The order was randomized (but the same for each user) to prevent the test being too repetitive for the user. At each stage the user is presented with a dialog box which prompts them for the type of click (**Figure 7.vi**). When the user has performed that click, the dialog box is dismissed and the next prompt it produced.

Additionally routines were written to provide some basic error catching which were not present in Li's Experiment, such as if an incomplete click was made (e.g. if a double click was made instead of a triple click) then the user would be prompted, after an appropriate wait, to redo the click. The program detected if the number of readings were too few for the current type of click, and if the user hadn't made any further clicks, after approximately 3 seconds they would be prompted to redo the click. Only the "re-done" click would be recorded but the clicks which were "re-done" would be recorded in the error field in the results.

At the end of the test, a dialog is presented thanking the user for their participation and returns them to the initial screen. The timings (in hexadecimal format) are then collated and written to a file in a comma separated value format for input into a spreadsheet package for analysis (the translated results are presented in Appendix 0).

The user is given an opportunity to exit the test at every step, if they choose to do so, then the incomplete results will be collated, and an error message of which point the test was ended at, noted.

8. Experimental Results and Discussion

Fifty users were surveyed; they were almost all in the age group 15 to 25 years old. Two users were in the oldest age group (between 45 to 50 years old), and two were between 25 and 30 years old. The majority of users were students resident at Halifax College at the University of York, and were tested in their kitchens, a large number of the students were personally known to me. The remaining users were students or staff in the department of Computer Science. Almost all the users placed themselves in the highest category of computer / mobile phone usage, with all the users using a computer / mobile phone at least once a day.

8.1. Single Click

	c (ms)
Mean	149.83
Median	125.00
Mode	116.00
Standard Deviation	96.04
Min	60.00
Max	1040.00

Table 8.i Statistical data collected for c

As the results above show (**Table 8.i**), the average timings seem to bear out the *Middleman* predictions from the Model Human Processor and also agree with Li's experimental results which show a mean of 146 ms for the age group of 10 to 50 year olds with an standard deviation of 96.5. The results for all the candidates' timings for c (10 timings each for 50 people) were then processed using SPSS to produce a frequency distribution graph (**Figure 8.i**).

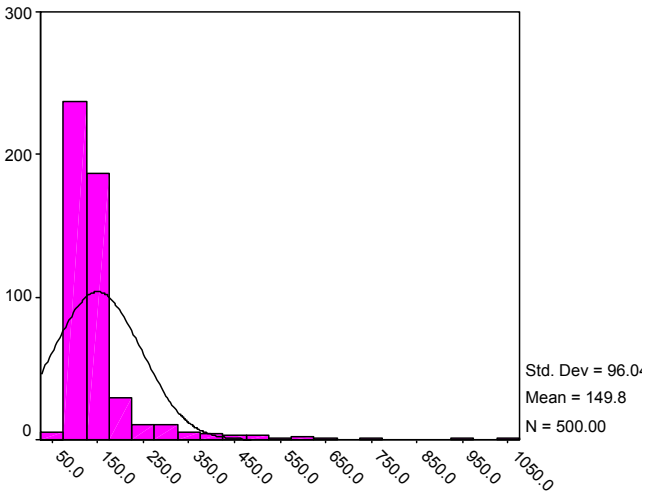


Figure 8.i Frequency Distribution for c

Looking further into the results the frequency distribution shows the majority of the timings in the *Fastman* region of approximately 60 ms. However the results seem to be distorted by a number of extremely slow timings (possibly due to some misinterpretation of the click to be performed). For the slower end of the timings, the Model Human Processor still seems to break down with a number of timings above the 200 ms predicted for *Slowman*. Due to the vast majority of users being part of the most frequent computer / mobile device usage group, unfamiliarity with task to be performed in unlikely, though ergonomic factors of the switch have not been taken into account previously and may have some affect on the timings.

8.2. Long Click

	l (ms)
Mean	669.54
Median	599.50
Mode	404.00
Standard Deviation	482.67
Min	67.00
Max	8357.00

Table 8.ii Statistical data collected for l

In the case of the long click, the Model Human Processor doesn't really apply; as there is no feedback from the device, it was entirely subjective and was when the user "felt" that they had

pressed the button long enough. The results do bear some similarity with the results from the age group of 10 to 20 year olds in Li's experiment (which show a mean of 860 ms); the majority of users in our study were in this age range also.

There were quite a few errors made in the recording of this measurement; many users tried to complete the test as fast as possible and mistakenly made a single click instead of a long click and this is shown by the skew in the results towards lower times. There was also one case where the user appeared to be waiting for some sort of prompt and just kept on holding the button (this is the timing recorded at 8 seconds). This is displayed in the frequency distribution graph, **Figure 8.ii**.

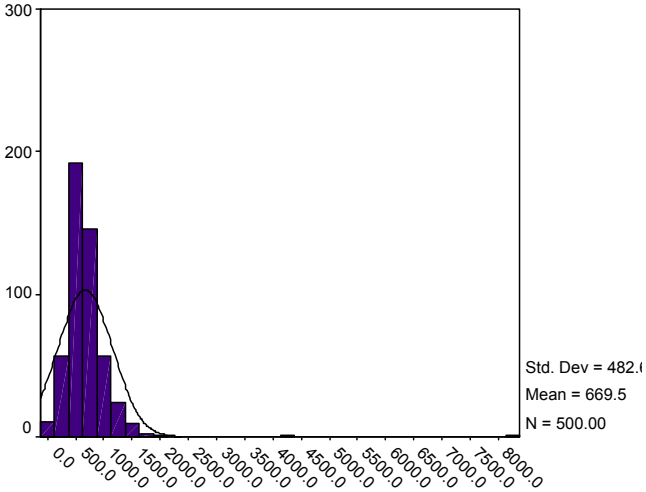


Figure 8.ii Frequency Distribution for l

8.3. Double Click

	d1 (ms)	d2 (ms)	d3 (ms)	d1+d2+d3 (ms)
Mean	139.47	117.47	123.91	380.84
Median	120.00	99.00	116.00	337.50
Mode	103.00	90.00	101.00	313.00
Standard Deviation	91.88	62.59	55.10	142.12
Min	43.00	54.00	62.00	220.00
Max	877.00	651.00	833.00	1240.00

Table 8.iii Statistical data collected for d1, d2, d3, d4 and d5

Again the results are in line with both Model Human Processor predictions and Li's results for the 10 to 50 year age group. The averages are slightly faster than the *Middleman* prediction of 140ms, possibly due to the number of users who would be categorised as in the *Fastman* performance group but agree more with Li's means, for the age group 10 to 50 year olds, which are shown in **Table 8.iv**. Li also had similar standard deviations.

	d1 (ms)	d2 (ms)	d3 (ms)
Mean	117.5	127.75	116.75
Standard Deviation	69	73.5	58.5

Table 8.iv Li's experimental data for double clicks in the age group 10 to 50 year olds

The fastest times in our experiment are in line with *Fastman* prediction, with very few results outside the *Slowman* prediction of 200ms. The results for ***d1***, ***d2***, and ***d3*** are all similar with a similar spread of results. The only minor difference being that of the gap between the clicks (***d2***) is slightly smaller than the duration of the clicks themselves. The clicks ***d1*** and ***d3*** are not as symmetric as in Li's experiment. With the values for d1 being slightly higher than those for ***d3***.

This is shown in the frequency distribution graphs for ***d1***, ***d2*** and ***d3*** in Appendix B.

8.4. Triple Click

	t1 (ms)	t2 (ms)	t3 (ms)	t4 (ms)	t5 (ms)	t1+t2+t3+t4+t5 (ms)
Mean	130.35	123.40	115.12	122.76	121.17	612.81
Median	116.50	106.00	106.00	103.00	115.00	550.50
Mode	112.00	81.00	97.00	99.00	100.00	506.00
Standard Deviation	68.88	62.77	39.66	87.47	37.40	194.80
Min	57.00	63.00	48.00	47.00	68.00	383.00
Max	724.00	646.00	547.00	1294.00	519.00	2058.00

Table 8.v Statistical data collected for t1, t2, t3, t4 and t5

Similarly for the triple click the average timings fit both the model human processor and Li's timing data in the same age range. The values obtained for ***t1***, ***t2***, ***t3***, ***t4*** and ***t5*** are reasonably symmetric with a skew towards timings expected of the *Fastman* model. The frequency distributions (in Appendix B) show the same patterns as we've seen earlier for the double click.

	t1 (ms)	t2 (ms)	t3 (ms)	t4 (ms)	t5 (ms)
Mean	119.3	135	109.75	145	117.25
Standard Deviation	62.75	91.25	56.25	102	58.75

Table 8.vi Li's experimental data for triple clicks in the age group 10 to 50 year olds

Direct comparison with Li's results (**Table 8.vi**) show Li's timings to have smaller values for the lengths of depression of the switch (***t1***, ***t3***, ***t5***) but higher means for the "gaps" between presses (***t2***, ***t4***). This could be due to the differing actions of the switches, an investigation into how different switches affect timings may shed some light on the difference in the results.

Additionally the symmetry of the timings is not quite as in Li's experiment, with the first value (***t1***) being higher than those of subsequent clicks, ***t3*** and ***t5***, (this pattern was also noted in the timings for the double click), though the timings ***t3*** and ***t5*** were symmetric, as were the timings for the "gaps", ***t2*** and ***t4***.

8.5. Errors

The errors for the user group were minimal, with only 10 clicks having to be redone out of 2000. There was no discernible pattern as to the types of clicks which this affected, or the types of users that made these mistakes.

9. The Effect of Different Switches on the Timings

One factor that may affect the timing is the type of switch used. In this experiment a small micro-switch was chosen with a black plastic cap. The amount of travel for this switch was small but could be made smaller. If different switches were used, with varying amount of travel, there may be some pattern to the variations in timings obtained. Additionally if different buttons were used, we may want to investigate the difference when different fingers are used to press the buttons. This experiment only really allowed for thumb presses, but changing the switch to mouse buttons, for example, may allow us to investigate different finger presses.

Although out of the scope of this project, as a few different switches were made before the experiment started, I chose to obtain some preliminary data, using these switches and a small sample size of ten people (due to time constraints) each performing the test a separate time on each switch (i.e. there were 100 values for each parameter, c, l, etc. we were investigating). All the users had previously performed the experiment and they were familiar with what was expected of them during the test.

Four switches were used to test what effect different switches would have on the timings. The switches were:

- A, the black switch used previously. (**Figure 7.i**)
- B, a left mouse button from a Logitech mouse. (**Figure 9.i**)
- C, a large Red push button switch. (**Figure 9.ii**)
- D, a micro-switch with a small white plastic cap. (**Figure 9.iii**)

These switches all had varying amounts of travel and resistance and all “felt” substantially different. In addition all the switches had different levels of feedback for when the button was sufficiently depressed. The switches and the results obtained from them are discussed in detail in the sections following.

9.1. Switch A

This black switch (pictured in **Figure 7.i**) was used with the new sample of participants. It offered the joint largest travel with Switch C, but offered less resistance than Switches C and D (however this resistance increased the further the switch was depressed).

	c	l	d1	d2	d3	t1	t2	t3	t4	t5
Mean	123.17	574.95	118.79	92.39	111.43	109.03	96.31	97.64	92.03	105.68
Median	114.00	572.00	110.00	86.50	110.50	107.00	87.50	97.50	85.00	105.00
Mode	117.00	460.00	115.00	80.00	107.00	113.00	86.00	104.00	73.00	105.00
Standard Deviation	53.50	191.52	53.40	23.84	14.93	20.27	33.97	13.27	34.51	14.13
Min	71.00	84.00	68.00	52.00	88.00	75.00	58.00	67.00	45.00	68.00
Max	527.00	1247.00	474.00	251.00	178.00	234.00	310.00	134.00	302.00	152.00

Table 9.i Statistical results for Switch A (times in ms)

As this was the switch used for the previous experiment we would expect to see similar results, however with the sample size being only ten people we've got much smaller standard deviations and results which closer approximate normal distribution; in addition to this we have the means being lower, indicating that the users were between the *Fastman* and *Middleman* models.

9.2. Switch B

The left mouse button from a Logitech mouse was used as a switch. Of the four switches used it offered the smallest travel and the least resistance.

Figure 9.i The mouse used for Switch B

	c	l	d1	d2	d3	t1	t2	t3	t4	t5
Mean	93.14	709.59	81.14	87.41	99.20	86.06	90.06	88.22	87.72	90.50
Median	85.50	712.50	80.00	82.00	98.00	77.00	86.50	86.50	80.00	89.50
Mode	88.00	703.00	84.00	89.00	98.00	73.00	85.00	93.00	80.00	92.00
Standard Deviation	41.23	249.74	21.06	49.40	18.80	58.71	31.70	15.77	53.11	14.49
Min	47.00	34.00	47.00	49.00	61.00	40.00	57.00	51.00	55.00	66.00
Max	322.00	1530.00	234.00	531.00	188.00	638.00	370.00	181.00	573.00	154.00

Table 9.ii Statistical results for Switch B (times in ms)

The mean values for the timings were lower across the board (apart from the timing for the long switch, ***l*** – this being a more subjective timing than the others). This could be due to the fact that there is less distance for the finger to travel or due to the fact that there is little force opposing the movement; additionally the index finger was used in all the cases to perform this click, whereas the thumb was used in the case of all the other switches. Interestingly the long press came very close to a normal distribution, (and much closer than in any of the other cases) but this could just be down to the small sample size used.

9.3. Switch C

This switch offered the most resistance and the most travel (joint with ***Switch A***). Additionally the action had very little to indicate when a connection had been made, with the button having to be fully depressed to be certain of a completed connection.

Figure 9.ii Switch C

	c	l	d1	d2	d3	t1	t2	t3	t4	t5
Mean	117.98	535.07	102.83	122.79	104.00	99.83	105.91	93.42	98.95	99.36
Median	106.50	515.50	97.50	103.00	102.50	98.00	104.50	90.00	99.00	96.50
Mode	138.00	425.00	87.00	94.00	100.00	91.00	109.00	99.00	95.00	99.00
Standard Deviation	50.98	173.67	43.83	96.51	27.08	21.77	16.25	19.75	18.44	23.77
Min	54.00	55.00	32.00	44.00	35.00	54.00	70.00	49.00	37.00	48.00
Max	419.00	994.00	337.00	843.00	200.00	164.00	148.00	172.00	166.00	170.00

Table 9.iii Statistical results for Switch C (times in ms)

Here, the results are slightly lower than those for the black switch, but still quite similar. Additionally the majority of the timings seemed to obey a normal distribution.

9.4. Switch D

The action of this switch was quite different to the others; with the white switch there was only felt to be a constant initial resistance, unlike ***Switch C*** where the resistance increased the further the switch was pressed. Additionally a positive click was heard when the button was sufficiently depressed.

Figure 9.iii Switch D

	c	l	d1	d2	d3	t1	t2	t3	t4	t5
Mean	138.78	639.58	116.45	73.89	115.13	118.47	74.41	105.58	77.57	110.70
Median	126.00	630.50	112.00	68.00	111.00	116.00	73.00	104.00	74.00	108.00

Mode	123.00	456.00	112.00	68.00	104.00	126.00	89.00	94.00	75.00	108.00
Standard Deviation	46.54	208.77	19.30	27.01	20.43	18.47	15.97	18.71	40.58	21.88
Min	92.00	61.00	69.00	48.00	42.00	75.00	46.00	47.00	40.00	72.00
Max	363.00	1293.00	184.00	295.00	176.00	204.00	123.00	157.00	445.00	184.00

Table 9.iv Statistical results for Switch D (times in ms)

Interestingly the results show extremely short mean times for the "gaps" between the clicks (***d2***, ***t2***, ***t4***); this could be due to a difference in the internals of how the switch operates, or the action performed by the user in depressing and releasing the switch.

9.5. Discussion

Switch	c	l	d1	d2	d3	t1	t2	t3	t4	t5
Switch A	123.17	574.95	118.79	92.39	111.43	109.03	96.31	97.64	92.03	105.68
Switch B	93.14	709.59	81.14	87.41	99.20	86.06	90.06	88.22	87.72	90.50
Switch C	117.98	535.07	102.83	122.79	104.00	99.83	105.91	93.42	98.95	99.36
Switch D	138.78	639.58	116.45	73.89	115.13	118.47	74.41	105.58	77.57	110.70

Table 9.v Comparison of means for the different switches (times in ms)

As you can see from the above comparison table (**Table 9.v**) there definitely seems to be some factors involving either, travel, resistance or the use of different fingers, which affect the timings obtained, a further study separating all these variables may elucidate which ones which are most important.

10. Conclusions

The purpose of this project was to obtain precise timing data for the use of a single button, as such it is important to compare the results with those collected in Li's project using a supposedly inaccurate software method. It is interesting to note that the results approximately agree despite the quoted accuracy of Li's software method of 50ms, which may suggest that the software system is either more sensitive than its baseline error suggests, or possibly that it "over-counts" as much as it "under-counts" and so the errors average out.

To provide guidelines as to the suitable timings for each of the parameters we were investigating (***c***, ***l***, ***d1***, ***d2***, ***d3***, ***t1***, ***t2***, ***t3***, ***t4***, ***t5***), we need to specify a level of acceptable error (i.e. a percentage of clicks which would be wrongly classified from our data, and our cut-offs). Previously an error rate of 40% was suggested for Tuffin's investigation into error rates [Tuffin, 2001]; this would provide guidelines for the different values as displayed in **Table 10.i**. However given that the population of this experiment is quite homogenous (unlike Li's experiment, upon which Tuffin's was based) smaller error rates may be more suitable.

Error Rate	c	l	d1	d2	d3	t1	t2	t3	t4	t5
40%	< 133	> 539	< 128	< 106	< 121	< 123	< 118	< 114	< 112	< 121
20%	< 159	> 411	< 150	< 133	< 137	< 142	< 141	< 130	< 132	< 138
10%	< 197	> 344	< 183	< 164	< 153	< 162	< 177	< 150	< 173	< 156

Table 10.i Suggested values for our parameters, from different error rates (times in ms)

An investigation on error rates similar to that conducted by Tuffin [2001], using the new test and equipment, should be used to verify that those error rates occur from the suggested values.

During the course of the project more questions raised themselves; these will be discussed in the next sections.

10.1. Further Work

10.1.1. Problems with the current experiment

One major problem still outstanding from the previous project was the ambiguity of the definition for the long click. In most portable systems there is feedback (e.g. auditory or visual feedback in terms of beeps or confirmation screens) for the single, double and triple clicks the user was

given feedback as the prompt for those clicks disappeared. For the long click there was no visual feedback as, clearly, for feedback to be given, the timing for this click needs to be known in advance, but it is exactly this timing we are trying to investigate.

An investigation similar to Tuffin's [2001], but using the new apparatus, may be able to give clearer timing data on the long click, but it remains to be seen whether it is better to define the long click in terms of the single click, i.e. anything longer than the range allowed for a single click would be classified as a long click.

Additionally, the test allowed no time for between the prompts; this led, in some cases, to errors caused by clicks "carrying over" to the next part of the test. For example if the user was prompted for a double click and made a triple click, due to the timing of the program, the last part of the triple click would be captured as part of the next click. This could cause errors to carry forward (but these cases were easy to spot and the users were asked to let the test prompt them to remake the current click; this would "reset" the buffers).

If a gap was placed between the tests, some of these errors would be caught automatically by routines already present. However some of the errors were caused due to users trying to complete the test in the quickest time, perhaps aided by the fact that there was no gap between the prompts. Changing the size of any gap may have some affect on reducing these types of errors.

Furthermore the switch timing, although more accurate than the previous software method, was only accurate to 10ms due to the de-bouncing of the switch.

10.1.2. Further investigation into the effect of different types of switches

The aim of this experiment was to obtain accurate timing results for various different types of button clicks. However the relationship between the timings and various factors were not investigated fully. A more detailed experiment than my second experiment (Section 7) covering a larger range of switches could also involve other ergonomic factors for the switch such as texture. When using multiple switches a set of different textures or shapes could be used for each switch. "A set of in-confusable shapes have been produced by different investigators over the years (for example, Hunt, 1953; Jenkins, 1947; Moore, 1974)" [Oborne, 1982].

10.1.3. Single button usage for differently able users

This experiment was designed to find optimal timings for able bodied candidates in a very restrictive age group. *"Such users are only one point on a wide and varied scale of physical capabilities"* [Keates et al., 2002]. One would expect the timings to vary considerably for candidates of different abilities. Keates et al. [2002] attempted to quantify where those differences occurred in the interaction cycle based on the Model Human Processor [Card et al., 1983]. A repeat of the experiment using a wider range of candidates of differing abilities could be based on Keates et al.'s revised models.

10.1.4. Combinations of different types of presses

There are also other types of button press we did not investigate, such as different lengths of the long click for repeated actions. These may be investigated in context to actually operations. An experiment designed within a set context using the data from this experiment as a base, may provide more useful results, for example testing a specific arrangement of buttons for set tasks. In many cases where the number of different functions for a single button need to be maximised (e.g. text input on mobile phones) a combination of different button presses are used. These combinations were not investigated, but may affect the timings obtained; additionally different combinations may affect the gaps needed between button presses and error rates for different presses. For example, if a double click were to be made following a single click (or vice versa) the gap between the clicks would have to be greater than the the value of $d2$ to prevent the click registering as a triple click

10.1.5. Relevance of the single button press

One final criticism of this project could be the relevance of a single button press. There have been advancements in the field of human interface design; many of these are to do with maximising the functionality of the buttons present. For example most modern mobile phones use a system of "soft buttons", where the buttons have no fixed meaning, but change depending on what screen is currently displayed. Additionally on some systems the users themselves can set the desired timings for double clicks (for example on Microsoft Windows), this could easily be extended for other types of clicks making the system respond optimally after an initial configuration period.

11. References

Axelson, 2000 — J. Axelson, *"USB Complete: everything you need to develop custom USB peripherals"*, Madison: Lakeview Research, 2001

BBC News, 2002a — BBC News, *"Mobile Keypad Reinvented"*, BBC News, May 2002 <http://news.bbc.co.uk/1/hi/sci/tech/1990855.stm>

BBC News, 2002b — BBC News, *"Mobile Keypad Gets Real"*, BBC News, November 2002 <http://news.bbc.co.uk/1/hi/technology/2504091.stm>

Card et al., 1983 — S.K. Card, A. Newell, T.P. Moran, *"The Psychology of Human-Computer Interaction"*, Hillsdale: Lawrence Erlbaum Associates, 1983

Edwards, 2002 — A. Edwards, *"Alistair Edwards' Project Suggestions"*, March 2002 <http://www-users.cs.york.ac.uk/%7Ealistair/projects/projects2002.html>

Edwards & Li, 2000 — A. Edwards, Y. Li, *"How Many Ways Can You Use One Button? Timing Data for Button Presses"*, York: University of York, 2000

Keates et al., 2002 — S. Keates, P. Langdon, P.J. Clarkson, P. Robinson, *"User Models and User Physical Capability"*, Cambridge: University of Cambridge, 2002

Li, 2000 — Y. Li, *"Timing Data for the Use of a Single Button"*, York: University of York, 2000

Oborne, 1982 — D.J.Oborne, *"Ergonomics at Work"*, Chichester: John Wiley, 1982

Price, 1983 — L. Price, *"Use of Mouse Buttons"*, Chicago: CHI'83 Proceedings, 1983

Tuffin, 2001 — J. Tuffin, *"Error Rates and User Perception in the Operation of a Single Button"*, York: University of York, 2001

Williams, 1983 — G. Williams, *"The Lisa Computer System"*, Byte Magazine, 1983

12. Bibliography

B. Huckle, *"The Man-Machine Interface"*, Carnforth: Savant Institute, 1981

W.H. Murray, C.H. Pappas, *"Visual C++ .NET : Complete Reference"*, Berkeley: Osborne, 2002

S. Oualline, *"Practical C Programming, 3rd Ed"*., Sebastapol: O'Reilly, 1997

J. Preece, *"Human Computer Interaction"*, Wokingham: Addison-Wesley, 1994

B. Stroustrup, *"The C++ Programming Language, 3rd Ed."*, New Jersey: Addison-Wesley, 1997

Appendixes

A. Results (translated from the hexadecimal output)

User Age	User Computer Usage	0t1	0t2	0t3	0t4	0t5	0l	0c	1c0	1t1	1t2	1t3	1t4	1t5	2c0
2	1	103	84	88	86	91	597	145	87	84	93	87	83	101	124
1	1	89	414	137	97	119	521	120	118	135	182	111	99	122	125
1	1	123	124	100	156	111	587	103	92	142	160	130	176	130	188
1	1	120	144	137	130	131	512	179	154	158	180	162	104	161	197
1	1	129	140	148	136	155	858	105	95	112	177	125	151	137	105
2	1	126	126	108	119	118	626	127	119	110	99	110	118	119	139
1	2	218	96	172	106	227	1027	108	133	108	119	186	105	157	113
1	2	145	261	128	235	116	786	113	116	141	202	136	164	131	108
1	1	191	159	130	114	181	389	193	112	114	116	118	90	137	139
1	1	132	73	100	86	128	674	129	121	99	91	111	92	117	118
2	4	99	127	100	112	96	830	140	123	106	120	102	92	105	128
1	1	118	113	119	136	134	684	148	122	121	115	133	119	143	156
1	1	106	157	135	120	108	349	100	151	150	185	141	111	122	149
1	1	126	90	113	109	114	688	97	97	98	103	97	86	101	110
2	1	148	78	132	105	127	857	128	125	121	129	128	128	146	131
1	1	118	123	95	103	96	338	154	111	110	122	79	104	120	130
1	2	116	86	139	102	133	891	153	121	134	96	113	118	83	115
1	3	135	107	150	115	151	1161	232	238	377	276	229	184	181	455
1	1	132	152	129	142	149	1267	318	90	116	137	123	121	111	131
1	2	117	77	126	101	140	653	101	102	80	89	77	81	97	118
1	1	147	127	155	132	138	974	131	147	148	164	138	172	158	143
1	1	112	263	145	157	138	500	131	609	459	140	306	74	259	605
1	1	369	133	204	132	197	679	119	124	127	118	91	117	151	146
1	1	152	130	167	138	207	739	168	177	133	110	130	100	164	186
1	1	100	96	132	107	136	456	163	181	164	121	125	109	136	170
7	1	70	137	71	121	121	704	91	132	106	249	142	690	95	338
7	1	207	242	298	251	335	1032	305	303	226	298	234	305	219	245
2	1	108	113	113	114	131	739	135	124	110	120	90	113	120	128
2	1	145	97	175	92	186	859	125	120	163	135	175	98	176	169
2	1	71	146	48	179	109	632	99	110	106	110	101	111	104	99
1	1	121	79	91	83	86	385	105	110	91	90	95	83	105	119
1	1	286	76	142	130	131	372	114	98	92	68	106	47	94	100
1	1	109	75	96	88	79	384	141	140	112	101	101	99	112	113
1	1	99	66	78	71	70	828	118	112	102	78	91	77	94	98
2	1	121	63	80	56	113	620	79	93	101	79	94	76	106	100
1	1	147	153	122	164	129	1014	126	112	123	172	115	126	106	114
1	2	121	195	116	342	116	897	148	146	174	156	150	180	148	143
1	1	134	87	97	90	101	660	123	103	130	78	114	74	114	134
1	1	133	86	105	134	109	1026	122	114	109	86	106	91	111	108
2	1	93	78	108	65	128	753	97	94	99	99	92	81	104	88
2	1	171	262	197	109	177	552	102	80	94	91	77	112	89	81
3	1	123	76	109	82	108	398	103	106	116	68	102	79	100	126
1	1	117	69	93	75	104	424	114	116	90	77	111	74	114	106
1	1	95	95	117	64	103	396	107	112	92	89	77	91	89	99
1	1	116	79	90	73	101	380	108	144	102	72	95	79	107	78
1	1	101	78	81	70	103	372	106	115	117	66	95	72	114	110
2	2	138	67	82	63	117	467	152	94	154	69	122	84	132	121
3	1	142	63	124	63	139	414	103	93	82	81	89	69	93	97
3	1	99	81	94	79	94	391	122	115	98	76	100	74	107	101
1	1	112	81	88	68	120	381	126	98	128	76	97	69	116	113

0d1	0d2	0d3	3c0	2t1	2t2	2t3	2t4	2t5	1l0	1d1	1d2	1d3	2d1	2d2	2d3	3t1	3t2	3t3	3t4	3t5
78	95	77	115	103	95	85	89	104	537	112	74	115	94	88	101	83	97	94	87	104
115	108	125	129	146	100	126	95	130	128	304	113	126	94	111	97	114	118	89	119	98
128	194	133	174	130	224	121	243	136	676	159	174	138	172	173	153	159	213	135	228	149
151	125	153	144	139	148	117	121	138	163	109	217	141	147	132	138	140	129	152	133	155
129	110	135	130	113	93	96	99	127	376	129	100	131	125	95	123	100	113	104	118	121
136	100	112	119	143	229	138	812	124	762	134	111	127	155	94	152	140	94	114	119	123
143	108	134	135	112	89	110	96	132	540	157	96	127	142	127	150	135	83	114	107	122
115	135	122	133	124	135	119	202	135	781	142	192	128	140	170	134	122	196	119	200	116
138	91	133	177	150	77	381	1294	156	130	159	150	137	158	132	125	136	129	114	125	113
120	96	97	110	102	102	97	95	109	690	116	120	101	101	110	99	116	120	96	115	112
98	90	119	103	111	107	119	85	93	547	95	92	103	102	87	103	79	109	110	87	98
132	116	139	156	141	114	137	117	157	538	141	115	158	143	126	152	115	134	114	149	111
188	108	118	171	159	118	116	177	127	575	152	133	125	165	119	132	132	146	121	129	133
111	87	115	119	98	98	101	105	114	486	105	101	110	75	89	104	86	94	117	116	133
113	89	127	114	168	146	166	130	121	8357	75	101	69	3094	103	125	79	124	82	112	111
111	110	105	127	115	123	86	128	112	309	102	110	105	113	114	103	121	124	75	117	123
104	88	101	107	94	92	114	79	121	492	75	99	117	107	82	126	125	105	96	89	130
121	120	139	254	104	128	169	91	170	941	151	93	137	179	82	157	178	121	151	77	186
118	106	113	113	120	142	78	115	93	4162	84	112	127	167	143	139	125	121	97	128	111
67	67	91	99	80	84	88	62	91	552	84	73	94	76	76	97	75	81	93	68	113
168	159	148	182	170	174	145	191	158	805	154	193	141	157	217	154	150	205	133	170	130
138	99	63	555	698	139	114	112	97	657	448	140	74	964	122	69	500	135	96	122	78
100	138	148	166	105	125	135	127	172	620	118	219	145	126	168	123	124	151	135	216	159
138	141	133	128	134	143	92	99	127	197	189	127	146	136	137	172	172	135	170	96	204
138	97	119	156	123	126	114	114	118	1289	129	105	134	129	100	132	127	94	123	104	124
125	80	113	126	123	92	105	79	111	723	123	82	110	122	90	118	123	90	91	97	118
192	185	240	229	181	214	186	254	220	982	232	260	269	234	239	191	195	239	179	308	229
124	117	123	121	105	97	114	120	124	657	106	140	120	110	107	124	115	124	117	125	133
146	108	176	169	157	128	126	118	168	943	153	130	163	142	127	170	157	135	131	116	152
106	63	109	81	88	122	66	112	106	608	106	70	103	115	71	108	107	135	79	106	105
97	98	99	97	95	102	83	101	90	365	100	94	86	87	120	86	95	109	83	93	93
97	67	108	99	79	77	93	425	102	475	93	66	110	101	74	112	93	79	98	68	113
97	84	107	115	136	91	106	75	106	689	106	99	108	125	86	102	103	95	101	112	76
110	81	109	110	99	90	85	98	73	448	113	74	95	103	68	122	116	88	141	136	519
86	91	93	105	94	81	100	67	104	581	109	79	105	99	73	106	117	78	101	75	98
129	111	103	200	131	227	104	325	144	1126	127	107	113	136	109	114	126	224	124	186	82
157	142	135	155	138	118	117	173	130	129	148	105	122	116	103	111	124	135	155	166	136
118	75	113	138	100	97	104	99	104	493	122	92	126	101	89	123	99	84	118	108	89
118	85	113	137	116	81	97	104	87	1089	103	91	102	121	72	110	111	103	100	106	104
92	94	99	104	97	82	78	81	93	690	91	98	91	98	75	100	104	86	86	96	83
77	104	86	93	90	97	83	104	79	715	105	89	98	158	95	96	112	82	85	74	91
134	106	87	128	120	96	100	102	96	651	101	85	122	101	108	103	121	113	83	131	89
91	86	104	86	100	95	87	81	97	540	89	89	93	72	99	97	94	95	93	95	101
106	87	90	113	99	98	85	86	101	101	143	74	118	205	84	132	129	95	96	76	115
102	81	98	90	101	88	92	89	90	303	101	88	89	86	87	103	93	76	105	99	74
100	88	97	109	100	85	93	95	107	398	92	83	110	115	84	98	98	81	83	83	103
104	71	91	104	96	83	93	71	106	437	160	97	126	92	75	103	120	76	97	69	105
70	87	79	92	76	94	64	94	74	344	80	92	81	81	103	71	74	107	62	95	76
92	67	108	91	97	93	78	95	95	473	119	76	124	98	83	123	113	84	103	82	103
93	88	104	193	112	84	100	74	117	389	126	109	126	99	76	109	99	81	100	85	99

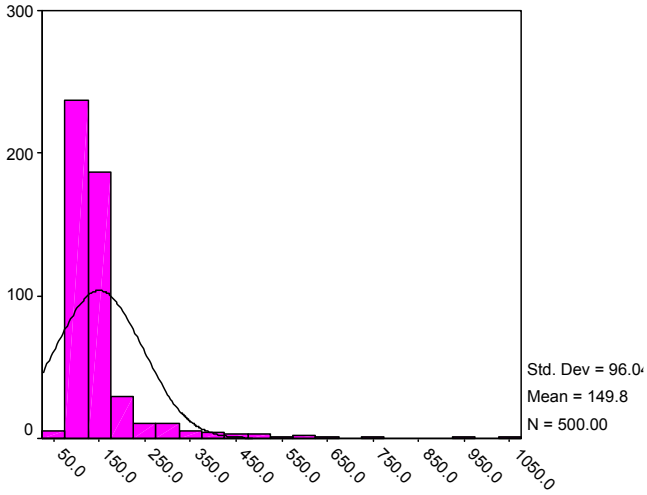
3d1	3d2	3d3	4c0	4d1	4d2	4d3	5c0	4t1	4t2	4t3	4t4	4t5	5t1	5t2	5t3	5t4	5t5	5d1	5d2	5d3
81	92	104	90	113	80	103	96	80	98	100	95	79	109	100	94	92	94	85	106	85
132	124	95	140	107	101	108	113	101	143	93	128	78	143	141	125	119	107	112	133	123
166	171	137	131	125	111	120	118	111	129	139	117	165	139	143	144	123	154	127	132	132
146	131	144	158	196	134	142	161	134	135	144	140	127	159	146	152	135	127	154	153	158
127	99	145	132	117	98	131	120	98	128	78	126	86	141	119	82	104	81	137	99	95
135	110	133	152	146	88	133	146	88	146	98	108	110	126	132	99	123	95	137	142	88
131	113	131	113	120	98	121	165	98	118	94	107	105	113	116	106	101	108	111	114	99
153	173	122	125	118	188	110	129	188	141	222	120	180	135	127	188	119	187	139	129	180
133	121	121	155	110	114	130	162	114	126	123	123	120	154	128	148	113	124	125	125	149
107	97	112	114	161	118	114	146	118	134	104	129	105	135	142	81	121	100	141	149	99
98	95	107	115	94	78	96	95	78	89	94	108	74	101	110	106	96	80	120	101	100
106	123	112	669	439	127	96	417	127	382	139	119	128	94	410	118	125	143	111	435	134
144	137	142	139	149	105	137	136	105	139	134	128	137	139	136	157	129	150	144	144	129
120	98	112	142	133	74	113	136	74	111	89	98	123	144	115	107	99	123	139	123	99
98	102	74	139	833	110	72	1040	110	139	106	80	100	84	142	100	84	96	103	104	81
146	99	117	117	109	115	113	137	115	123	103	109	97	118	130	113	97	100	128	105	100
100	97	122	131	170	152	115	141	152	136	84	91	71	123	94	89	108	80	107	129	499
163	91	177	190	159	92	168	189	92	137	96	112	85	138	107	106	117	81	130	113	109
131	98	125	152	113	87	121	164	87	109	162	102	139	115	131	165	82	169	134	128	154
92	75	110	306	76	73	88	65	73	67	86	83	76	85	82	83	73	80	93	71	85
139	133	110	123	143	195	132	151	195	159	132	131	169	134	161	141	122	179	144	148	153
479	116	91	513	422	96	66	481	96	468	129	240	134	94	367	113	83	114	83	443	116
123	196	150	159	142	162	135	219	162	110	150	128	110	162	91	142	127	131	138	119	181
220	243	167	158	131	161	82	167	161	143	280	165	151	125	180	154	130	149	198	117	153
138	92	144	146	117	110	119	147	110	115	103	114	93	108	113	101	118	92	127	117	101
140	107	120	143	128	95	118	124	95	113	97	84	100	108	129	100	107	106	105	122	96
198	192	223	207	217	276	209	215	276	183	242	208	257	219	219	295	213	277	223	220	246
118	136	124	159	123	117	118	123	117	122	126	115	112	123	128	113	130	126	130	119	78
144	108	171	150	163	108	155	184	108	123	142	128	126	153	135	126	131	112	170	133	119
118	95	106	108	113	75	104	129	75	99	150	85	107	105	89	108	103	117	94	92	77
88	99	106	86	82	115	81	112	101	88	98	87	99	94	92	102	99	84	92	90	106
76	91	97	288	92	91	102	69	68	88	80	64	94	74	81	92	74	92	61	87	79
122	86	121	129	105	80	128	257	104	87	95	79	108	92	98	97	98	89	100	98	116
90	89	101	120	91	79	91	92	94	68	81	80	93	91	95	68	91	81	73	74	76
106	75	98	199	103	79	84	84	97	83	83	82	86	82	266	87	76	101	104	81	98
118	156	94	128	114	314	136	144	124	132	96	179	121	132	111	105	156	94	136	113	90
121	120	128	198	97	156	120	154	115	136	121	153	100	111	124	110	148	109	154	118	130
120	114	119	122	143	90	140	131	117	131	102	103	103	117	523	125	99	113	90	90	111
95	85	99	83	103	86	104	116	117	118	112	97	103	117	90	100	107	86	144	79	114
163	101	109	129	94	68	110	90	80	80	73	81	88	69	85	72	86	76	87	82	91
86	74	90	98	76	87	88	85	108	86	86	89	68	83	92	86	91	88	156	93	83
117	89	107	136	109	104	108	100	119	107	91	100	92	79	95	84	105	83	84	104	114
105	84	118	143	96	134	119	108	116	78	103	79	102	107	82	110	85	121	210	160	111
84	100	101	90	107	83	113	173	90	92	91	82	121	87	88	98	80	103	129	338	137
95	94	95	95	104	84	99	103	89	93	85	94	79	113	73	93	88	101	71	86	85
82	97	97	134	92	91	94	116	79	93	86	87	100	111	78	87	98	89	111	88	117
102	74	112	111	112	62	99	113	113	95	91	79	99	123	83	95	77	101	117	74	102
103	86	88	101	71	88	88	124	96	83	88	87	105	101	86	87	93	97	97	96	107
105	78	111	133	88	84	103	116	100	92	92	86	111	97	99	90	85	98	86	99	99
93	87	105	123	122	77	102	109	101	77	95	86	99	91	79	93	90	94	92	85	91

6t1	6t2	6t3	6t4	6t5	2l0	3l0	6c0	4l0	7c0	6d1	6d2	6d3	7t1	7t2	7t3	7t4	7t5	5l0	7d1
91	98	92	91	97	569	638	110	605	148	83	85	97	124	75	103	89	119	532	127
122	341	126	87	110	129	143	158	156	193	188	441	139	130	116	112	202	141	149	121
146	154	116	160	146	450	577	107	725	116	109	112	123	148	188	122	153	129	632	133
113	155	158	145	155	591	684	332	642	185	226	142	157	136	145	135	124	161	127	124
81	115	92	121	97	416	429	140	415	253	124	87	129	119	86	103	93	119	570	131
135	92	113	91	125	476	430	111	487	184	130	104	129	112	92	129	96	139	473	146
115	116	80	101	119	892	910	126	786	133	127	99	108	112	95	103	102	124	1024	114
132	204	121	200	138	715	691	129	796	136	137	202	130	124	214	123	209	131	904	132
139	135	117	133	139	474	639	168	515	160	113	159	123	143	129	104	98	139	431	158
139	140	159	97	154	743	795	125	754	134	141	131	129	125	94	125	96	121	669	140
95	102	110	82	117	646	596	91	544	116	83	90	106	112	100	108	90	112	664	103
447	134	151	140	128	344	1591	408	434	946	447	236	82	425	121	99	137	109	536	282
122	140	120	117	119	502	563	155	698	160	736	139	163	825	172	105	132	110	642	396
114	101	96	116	146	149	837	116	752	171	139	87	134	143	88	123	100	131	489	88
112	73	98	103	101	94	723	111	952	100	99	82	101	132	111	108	112	100	864	114
114	92	127	226	112	290	346	110	394	172	193	107	126	101	102	123	88	146	365	134
136	76	123	85	128	509	565	108	304	100	393	82	153	142	89	111	83	118	759	122
120	97	126	111	128	749	902	139	880	138	178	93	148	159	104	114	100	162	1571	403
142	134	116	163	132	1044	1365	130	1246	128	107	114	117	106	319	150	190	161	1159	130
106	71	92	70	116	544	552	96	570	72	63	82	102	70	82	79	68	100	67	72
162	149	131	156	120	830	756	210	707	158	161	97	120	149	164	135	108	138	749	155
498	139	186	141	100	573	377	482	529	312	804	129	68	724	116	70	112	68	762	1250
145	152	114	305	187	561	656	120	665	165	152	178	172	142	161	114	181	153	694	193
157	125	119	137	110	551	668	123	628	130	365	149	129	164	144	150	132	202	595	172
136	105	108	91	130	566	648	146	763	150	135	74	131	106	106	109	80	124	531	130
126	98	96	99	121	772	1052	117	837	150	137	95	115	113	102	93	100	119	1015	114
226	274	202	270	210	920	851	370	742	386	217	242	212	196	214	190	203	197	724	212
112	71	111	76	109	232	497	133	807	136	115	90	120	101	106	115	117	112	846	93
132	132	132	125	169	1154	1092	147	900	181	143	131	121	130	129	113	114	149	974	134
88	161	103	124	95	953	770	116	822	138	132	54	115	103	95	97	107	87	736	80
103	81	94	86	90	424	541	90	475	107	234	114	100	110	95	100	93	101	418	98
105	77	106	60	102	1128	1012	65	854	83	75	79	103	118	66	93	71	112	668	109
108	101	104	107	81	478	781	118	655	119	126	86	115	135	72	110	100	102	875	126
70	92	65	85	71	660	792	108	1245	115	106	61	116	81	86	104	84	98	1311	132
90	92	97	70	100	1163	874	115	967	121	115	90	100	111	93	91	93	90	720	103
121	102	103	129	99	1123	963	116	783	121	497	130	115	135	97	114	111	118	865	120
119	127	126	156	116	1421	904	151	741	118	136	104	125	145	132	134	172	121	894	191
130	118	90	283	128	544	867	126	828	122	131	98	119	136	89	95	117	88	712	108
135	83	98	110	80	1050	1613	127	1661	118	107	94	110	97	94	97	95	100	891	93
58	86	78	90	80	967	1016	107	640	107	74	97	82	92	81	85	82	93	470	94
93	87	81	93	80	371	378	117	345	109	109	83	120	102	95	95	105	109	342	111
113	97	86	284	137	608	576	110	523	125	128	91	124	103	92	113	92	108	601	134
116	117	78	146	111	564	519	162	583	127	201	103	126	119	100	107	78	119	310	130
106	106	119	100	103	676	485	90	491	173	155	84	133	112	97	112	97	128	411	115
76	87	87	77	91	404	451	86	383	81	90	88	94	66	87	81	93	76	256	91
94	90	97	96	116	414	458	96	351	60	105	69	120	101	80	100	86	107	490	126
88	96	78	77	108	390	418	96	79	110	79	91	101	106	79	101	70	113	506	100
91	98	91	99	108	385	362	88	371	104	113	75	282	120	89	115	81	118	397	121
102	94	98	84	109	445	502	115	289	105	150	94	128	127	86	106	98	104	360	136
111	86	99	85	96	126	302	98	317	120	106	67	115	107	80	90	76	116	272	99

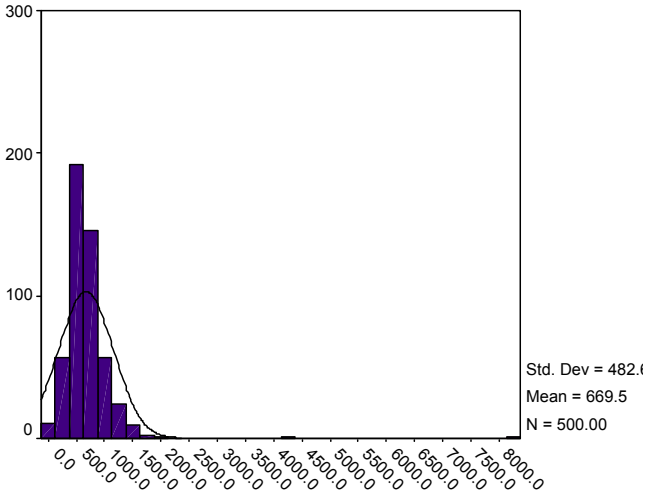
7d2	7d3	8d1	8d2	8d3	9d1	9d2	9d3	8t1	8t2	8t3	8t4	8t5	6l0	8c0	7l0	9t1	9t2	9t3	9t4	9t5
86	113	102	87	93	111	104	86	96	90	99	87	92	444	119	555	124	89	98	97	94
585	176	132	651	198	130	124	120	156	121	147	305	172	466	169	166	156	300	160	141	148
138	135	137	124	139	175	125	147	134	144	127	154	147	869	153	741	123	142	136	107	149
144	153	137	146	165	154	148	149	149	96	242	181	165	592	163	148	143	141	140	519	146
113	138	106	94	127	100	99	123	111	164	114	141	109	417	135	506	114	112	121	106	122
102	140	129	105	143	151	90	149	139	104	104	87	124	353	139	356	171	119	113	109	121
95	134	102	92	116	108	101	117	109	102	93	92	123	774	106	683	120	92	99	85	121
216	118	150	165	128	128	166	121	166	172	126	188	144	848	136	831	132	181	123	187	131
123	151	127	106	159	144	124	161	147	129	123	117	120	386	181	440	172	145	124	129	152
100	127	121	96	127	117	109	136	93	116	151	103	105	781	141	718	136	123	105	84	126
83	127	114	98	101	64	116	112	90	95	124	83	116	115	122	667	112	107	111	62	120
381	75	279	138	230	363	119	122	388	137	112	122	95	383	107	576	332	141	93	155	126
92	187	877	164	118	769	168	129	620	141	122	117	100	504	437	475	218	148	149	139	137
135	833	76	85	643	541	157	542	85	114	547	142	69	87	739	676	339	112	109	102	90
93	142	106	93	119	125	94	138	126	128	121	165	123	934	107	1126	112	146	107	140	131
91	115	183	327	136	152	389	120	137	89	108	102	133	284	153	357	154	78	121	83	143
84	114	119	97	100	120	74	114	135	84	116	58	142	481	116	889	139	76	121	72	120
106	152	120	106	150	157	86	143	137	99	122	113	141	1751	205	1358	134	127	93	144	155
124	134	117	124	115	133	153	111	108	113	117	127	108	1031	143	1352	107	133	117	94	128
78	101	43	80	97	85	88	105	110	131	105	90	120	84	150	525	57	100	81	83	101
162	126	159	149	113	144	232	140	153	210	121	217	133	827	359	806	136	223	148	236	153
95	76	462	114	131	412	120	70	448	276	109	113	145	487	319	392	282	114	76	120	82
144	188	165	140	150	171	143	173	145	137	147	147	160	813	179	806	146	114	131	127	156
142	147	158	158	182	173	146	110	166	125	123	115	147	642	184	570	157	284	154	116	186
97	121	134	95	143	210	115	112	110	114	90	95	126	642	150	739	131	113	108	104	130
95	103	123	88	116	115	101	123	135	91	96	98	114	838	96	819	121	81	104	115	126
223	211	200	274	210	217	260	253	207	303	209	309	226	979	330	894	214	214	228	231	236
93	107	100	93	120	98	97	111	119	94	97	87	119	833	86	976	129	89	109	75	131
125	121	124	118	154	125	100	171	120	133	115	108	159	869	138	889	105	146	135	106	153
81	98	135	74	102	107	89	101	132	101	73	70	121	638	116	541	137	86	101	103	98
84	107	119	76	109	118	96	98	107	87	97	104	85	438	100	447	101	92	111	91	111
85	90	90	85	97	103	77	101	89	74	101	81	93	585	101	545	100	87	94	79	113
98	102	119	90	117	105	84	107	78	109	91	96	98	1022	131	655	129	74	115	76	114
296	133	109	70	105	103	63	102	88	81	84	82	86	1162	122	1228	118	87	82	84	119
86	103	114	88	101	123	83	104	108	99	68	266	125	623	131	472	99	84	95	74	107
199	131	144	141	86	136	185	151	131	151	133	158	125	768	132	903	136	131	115	162	125
373	169	174	287	165	202	298	156	235	309	175	467	155	1254	394	1273	181	438	176	323	155
102	111	104	95	107	113	86	112	61	106	97	106	84	489	115	176	114	104	103	84	101
95	110	110	101	98	104	105	80	128	81	113	107	94	88	119	1402	106	109	98	100	102
84	107	108	92	117	91	83	125	99	88	103	95	107	558	90	485	132	79	101	90	97
105	100	117	68	124	100	87	113	108	85	101	91	105	574	124	409	114	82	104	88	118
81	126	127	100	106	115	92	113	134	87	120	86	117	554	92	572	92	165	92	94	116
79	122	103	90	116	111	75	107	111	87	99	89	290	487	121	404	101	93	90	152	93
86	128	110	80	133	126	87	111	162	132	91	115	84	476	140	413	119	646	126	157	123
346	62	79	90	81	75	93	91	80	93	87	95	89	445	97	324	94	81	98	83	102
89	116	103	90	105	104	93	100	78	262	119	89	123	453	105	342	139	111	87	93	118
90	110	95	107	88	93	90	102	102	99	95	82	125	567	153	526	123	100	94	89	115
76	132	103	93	120	127	85	122	201	106	117	104	134	528	167	126	178	107	127	99	136
146	113	122	77	140	109	72	116	168	68	125	65	138	131	227	480	163	65	155	97	165
80	102	98	82	97	86	85	94	105	71	101	81	104	94	111	610	123	185	103	595	142

8l0	9c0	9l0	Errors
571	122	444	
748	147	470	
708	104	827	
880	157	145	
503	129	759	
383	125	330	R:0.
819	122	665	
707	124	830	
449	137	537	
749	139	792	
800	135	814	
595	251	474	
404	426	390	
695	301	563	
786	142	919	R:19.
357	147	153	
686	100	915	
1213	203	949	
1448	110	1433	R:11.
619	114	554	
771	114	806	
421	288	389	
641	158	721	R:0.
717	164	598	
960	161	805	
815	113	1183	
708	274	755	
1186	125	596	
928	125	882	
704	112	1014	
372	95	377	
483	126	522	R:27.
539	139	1273	R:34.
1326	149	1507	R:28.
1094	100	1039	
670	170	166	
828	162	679	
968	128	780	
2056	120	1567	R:34.
414	121	481	
430	138	462	
677	135	574	
465	169	475	
404	141	525	
287	80	72	
424	130	590	R:4.
460	120	545	R:15.
488	116	400	
452	298	481	
471	133	471	

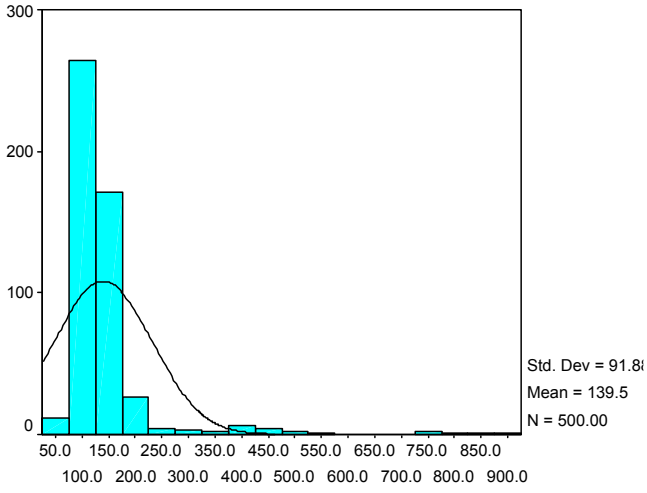
B. Frequency Distribution Graphs



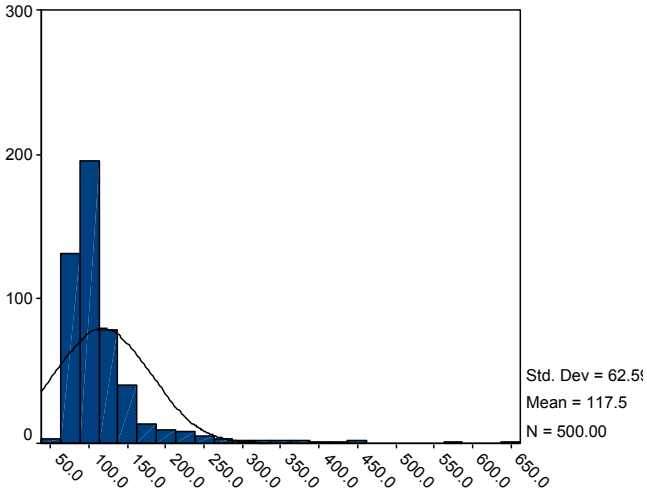
C



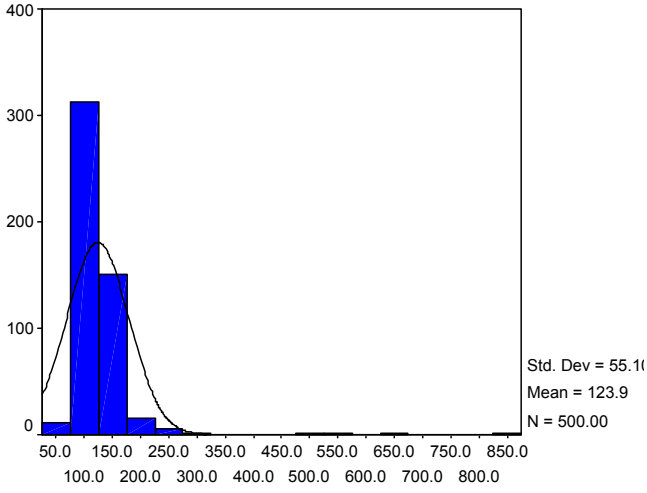
L



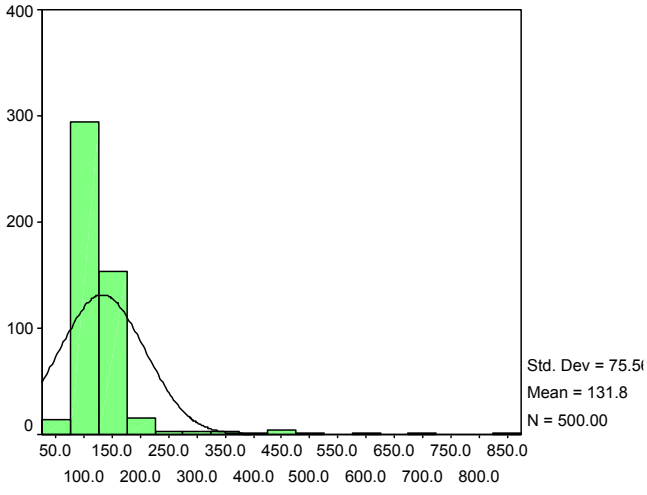
D1



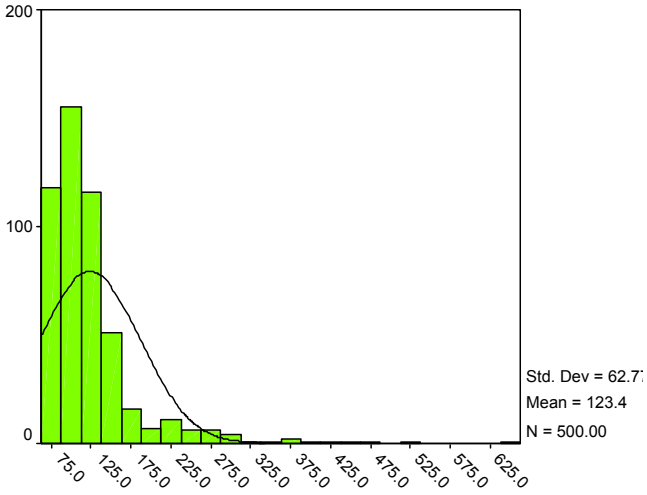
D2



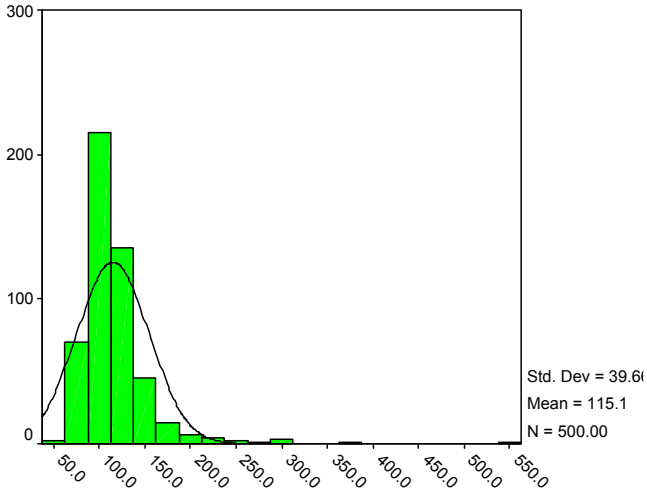
D3


400
300
200
100
0
50.0 150.0 250.0 350.0 450.0 550.0 650.0 750.0 850.0
100.0 200.0 300.0 400.0 500.0 600.0 700.0 800.0
Std. Dev = 75.5
Mean = 131.8
N = 500.00

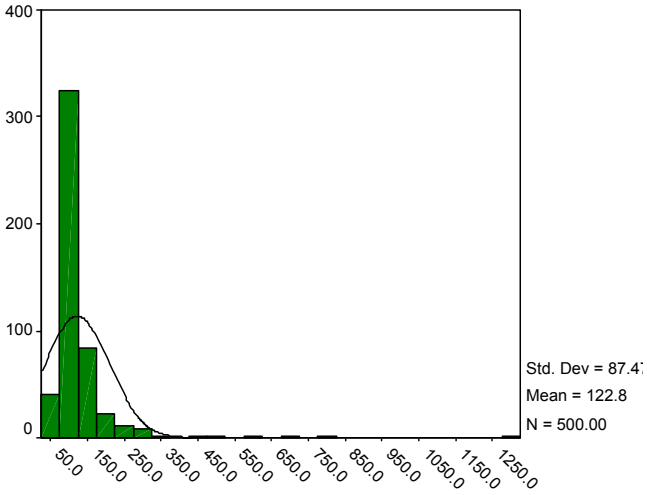
T1


200
100
0
75.0 125.0 175.0 225.0 275.0 325.0 375.0 425.0 475.0 525.0 575.0 625.0
Std. Dev = 62.7
Mean = 123.4
N = 500.00

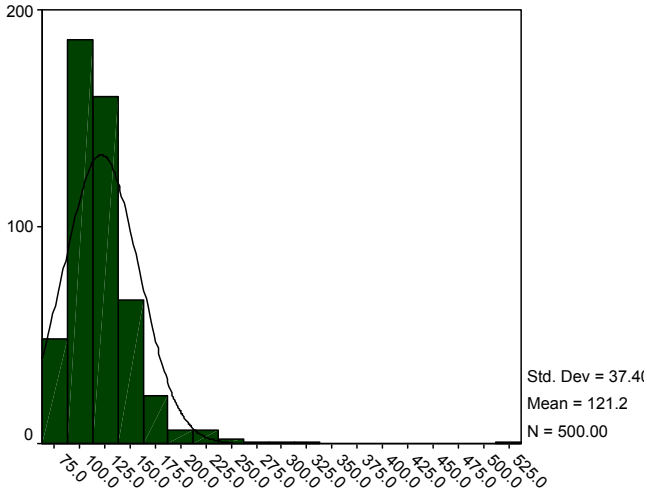
T2


300
200
100
0
50.0 100.0 150.0 200.0 250.0 300.0 350.0 400.0 450.0 500.0 550.0
Std. Dev = 39.6
Mean = 115.1
N = 500.00

T3


400
300
200
100
0
50.0 150.0 250.0 350.0 450.0 550.0 650.0 750.0 850.0 950.0 1050.0 1150.0 1250.0
Std. Dev = 87.4
Mean = 122.8
N = 500.00

T4


200
100
0
75.0 100.0 125.0 150.0 175.0 200.0 225.0 250.0 275.0 300.0 325.0 350.0 375.0 400.0 425.0 450.0 475.0 500.0 525.0
Std. Dev = 37.4
Mean = 121.2
N = 500.00

T5

C. User Instructions

About the Test

The experiment consists of prompts for different types of button presses. The four different kinds of button presses you will be asked to perform are detailed below:

Single Click

This is when the button is pushed down and then immediately released.

Long Click

This is when the button is pushed down, held in this position for an extended period of time and then released. The time taken should be long enough to distinguish it from a single-click but not so long as to induce impatience if someone were asked to repeat this action. The time for which the button is held down is determined by you.

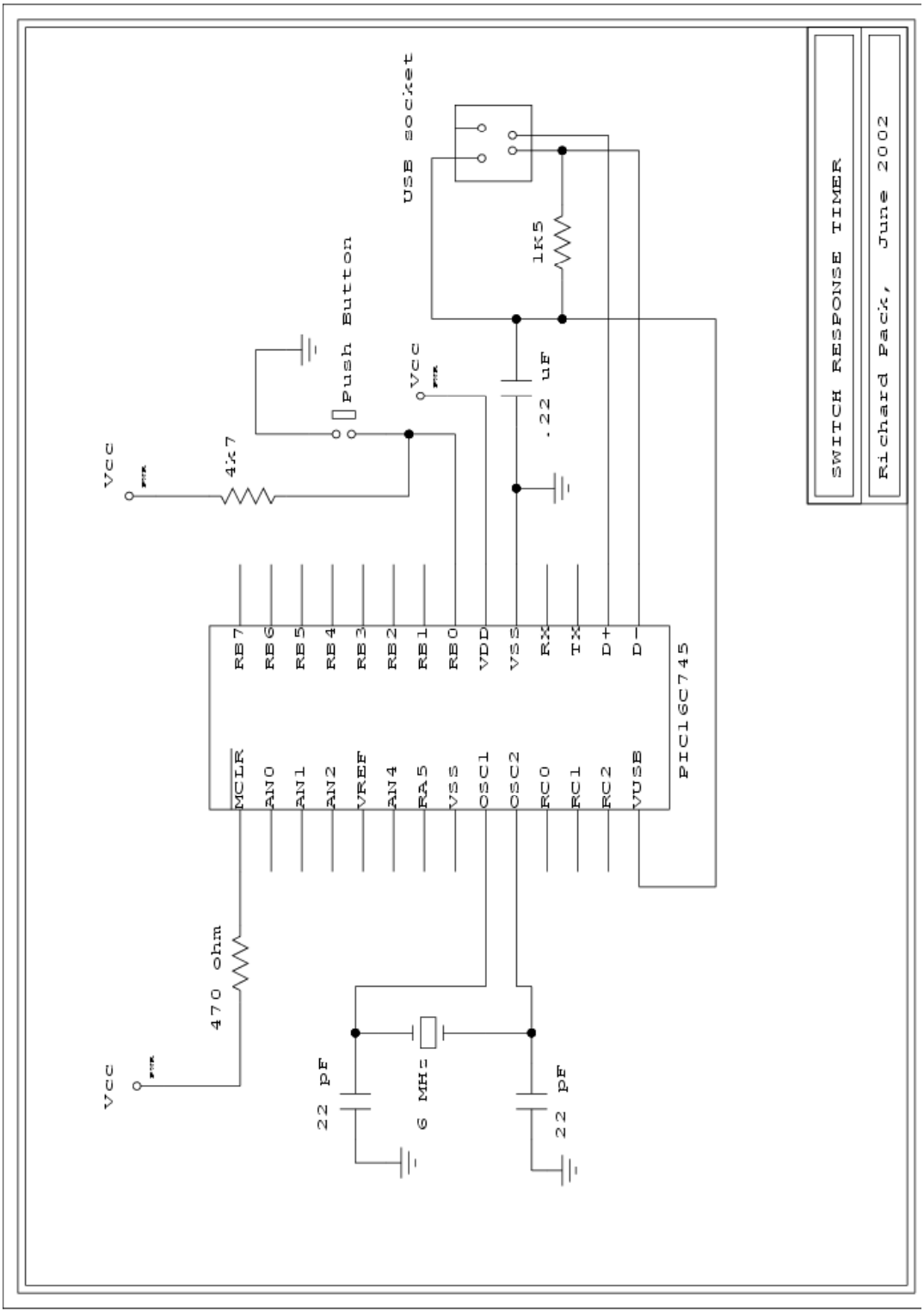
Double Click

This is the equivalent of two single clicks in rapid succession

Triple Click

This is the equivalent of three single clicks in rapid succession

D. Switch Schematic



E. Visual C++ Code

I. USB Switch Timer.h

```
// USB Switch Timer.h : main header file for the USB Switch Timer application
//

#pragma once

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols


// CUSBSwitchTimerApp:
// See USB Switch Timer.cpp for the implementation of this class
//

class CUSBSwitchTimerApp : public CWinApp
{
public:
	CUSBSwitchTimerApp();

// Overrides
	public:
	virtual BOOL InitInstance();

// Implementation
	DECLARE_MESSAGE_MAP()
};

extern CUSBSwitchTimerApp theApp;
```

II. USB Switch Timer.cpp

```
// USB Switch Timer.cpp : Defines the class behaviours for the application.
//

#include "stdafx.h"
#include "USB Switch Timer.h"
#include "USB Switch TimerDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CUSBSwitchTimerApp

BEGIN_MESSAGE_MAP(CUSBSwitchTimerApp, CWinApp)
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()


// CUSBSwitchTimerApp construction

CUSBSwitchTimerApp::CUSBSwitchTimerApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}


// The one and only CUSBSwitchTimerApp object

CUSBSwitchTimerApp theApp;


// CUSBSwitchTimerApp initialization

BOOL CUSBSwitchTimerApp::InitInstance()
{
	// InitCommonControls() is required on Windows XP if an application
	// manifest specifies use of ComCtl32.dll version 6 or later to enable
	// visual styles.  Otherwise, any window creation will fail.
	InitCommonControls();

	CWinApp::InitInstance();

	AfxEnableControlContainer();
```

```
	CUSBSwitchTimerDlg dlg;
	m_pMainWnd = &dlg;
	INT_PTR nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}
```

III. USB Switch TimerDlg.h

```
// USB Switch TimerDlg.h : header file
//

#pragma once

// CUSBSwitchTimerDlg dialog
class CUSBSwitchTimerDlg : public CDHtmlDialog
{
// Construction
public:
	CUSBSwitchTimerDlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	enum { IDD = IDD_USBSWITCHTIMER_DIALOG, IDH = IDR_HTML_USBSWITCHTIMER_DIALOG };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support

	HRESULT OnButtonOK(IHTMLElement *pElement);
	HRESULT OnButtonCancel(IHTMLElement *pElement);

// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	virtual BOOL OnInitDialog();
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
	DECLARE_DHTML_EVENT_MAP()
};
```

IV. USB Switch TimerDlg.cpp

```
// USB Switch TimerDlg.cpp : implementation file
//

#include "stdafx.h"
#include "USB Switch Timer.h"
#include "USB Switch TimerDlg.h"

#include "LegalDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// 1 or 0 depending on whether the legal agreement box is checked
int legal_agree;

// 0 (null) to 7 depending on what radio button the user selects for age
int user_age;

// 0 (null) to 7 depending on what radio button the user selects for age
int user_computer_usage;

// 1 or 0 depending on whether the test was finished or not
int unfinished=0;

// the test headings for the top of the csv file
CString                    test_headings[100]                    =                    {
        "0t1","0t2","0t3","0t4","0t5","0l0","0c0","1c0","1t1","1t2",

        "1t3","1t4","1t5","2c0","0d1","0d2","0d3","3c0","2t1","2t2",

        "2t3","2t4","2t5","1l0","1d1","1d2","1d3","2d1","2d2","2d3",

        "3t1","3t2","3t3","3t4","3t5","3d1","3d2","3d3","4c0","4d1",

        "4d2","4d3","5c0","4t1","4t2","4t3","4t4","4t5","5t1","5t2",

        "5t3","5t4","5t5","5d1","5d2","5d3","6t1","6t2","6t3","6t4",

        "6t5","2l0","3l0","6c0","4l0","7c0","6d1","6d2","6d3","7t1",

        "7t2","7t3","7t4","7t5","5l0","7d1","7d2","7d3","8d1","8d2",

        "8d3","9d1","9d2","9d3","8t1","8t2","8t3","8t4","8t5","6l0",
```

```
        "8c0","7l0","9t1","9t2","9t3","9t4","9t5","8l0","9c0","9l0"};

// int indicating the test position an unfinished test was halted at
int halt = 0;


// Cfile setup
CFile out_file;
CFileException fileException;
CFileStatus            f_status;

// CUSBSwitchTimerDlg dialog

BEGIN_DHTML_EVENT_MAP(CUSBSwitchTimerDlg)
        DHTML_EVENT_ONCLICK(_T("ButtonOK"), OnButtonOK)
        DHTML_EVENT_ONCLICK(_T("ButtonCancel"), OnButtonCancel)
END_DHTML_EVENT_MAP()


CUSBSwitchTimerDlg::CUSBSwitchTimerDlg(CWnd* pParent /*=NULL*/)
        : CDHtmlDialog(CUSBSwitchTimerDlg::IDD, CUSBSwitchTimerDlg::IDH, pParent)
{
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);

        //Open the file
        char* pszFileName = "./output.csv";
        bool file_new = FALSE;

        if( CFile::GetStatus( pszFileName, f_status ) )
        {

        // Open the file without the Create flag

        if(  !out_file.Open(  pszFileName,  CFile::modeCreate  |  CFile::modeNoTruncate  |
CFile::modeWrite, &fileException ) )
        {
                #ifdef _DEBUG
                afxDump << "File could not be opened " << fileException.m_cause << "\n";
                #endif
        }
        out_file.SeekToEnd();
        }
        else
        {
        if(  !out_file.Open(  pszFileName,  CFile::modeCreate  |  CFile::modeNoTruncate  |
CFile::modeWrite, &fileException ) )
        {
```

```
                #ifdef _DEBUG
                afxDump << "File could not be opened " << fileException.m_cause << "\n";
                #endif
        }
        out_file.SeekToEnd();

        // write heading to file if empty

        CString Headings;

        Headings += "User Age,User Computer Usage,";

        int i;
        for (i=0;i<100;i++)
        {
                Headings += test_headings[i];
                Headings += ",";
        }
        Headings += "Errors\n";
        out_file.Write(Headings,Headings.GetLength());
        }
}

void CUSBSwitchTimerDlg::DoDataExchange(CDataExchange* pDX)
{
        CDHtmlDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CUSBSwitchTimerDlg, CDHtmlDialog)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


// CUSBSwitchTimerDlg message handlers

BOOL CUSBSwitchTimerDlg::OnInitDialog()
{
        CDHtmlDialog::OnInitDialog();

        // Set the icon for this dialog.  The framework does this automatically
        //  when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);                        // Set big icon
        SetIcon(m_hIcon, FALSE);              // Set small icon

        ShowWindow(SW_MAXIMIZE);

        // TODO: Add extra initialization here
```

```
	return TRUE;  // return TRUE  unless you set the focus to a control
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CUSBSwitchTimerDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND,   reinterpret_cast<WPARAM>(dc.GetSafeHdc()),
0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDHtmlDialog::OnPaint();
	}
}

// The system calls this function to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CUSBSwitchTimerDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}

// if the New User button is pressed
HRESULT CUSBSwitchTimerDlg::OnButtonOK(IHTMLElement* /*pElement*/)
{
	// initialise all the variables
	legal_agree = 0;
	user_age = 0;
	user_computer_usage = 0;
	unfinished = 0;
```

```
        halt = 0;
        // call the legal dialog
        CLegalDlg Dlg;
        Dlg.DoModal();
        return S_OK;
}

// if the Quit button is pressed
HRESULT CUSBSwitchTimerDlg::OnButtonCancel(IHTMLElement* /*pElement*/)
{
        //Close file
        out_file.Close();
        OnCancel();
        return S_OK;
}
```

V. LegalDlg.h

```
// LegalDlg.h

#pragma once

// CLegalDlg dialog

class CLegalDlg : public CDHtmlDialog
{
        DECLARE_DYNCREATE(CLegalDlg)

public:
        CLegalDlg(CWnd* pParent = NULL);   // standard constructor
        virtual ~CLegalDlg();
// Overrides
        HRESULT OnButtonOK(IHTMLElement *pElement);
        HRESULT OnButtonCancel(IHTMLElement *pElement);

// Dialog Data
        enum { IDD = IDD_LEGAL, IDH = IDR_HTML_LEGALDLG };

protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual BOOL OnInitDialog();

        DECLARE_MESSAGE_MAP()
        DECLARE_DHTML_EVENT_MAP()
public:
        afx_msg void OnBnClickedLegalok();
        afx_msg void OnBnClickedLegalcancel();
        afx_msg void OnBnClickedLegalagree();
        afx_msg void OnEnChangeLegal();
};
```

VI. LegalDlg.cpp

```
// LegalDlg.cpp : implementation file
// This is the file which controls the legal agreement dialog

#include "stdafx.h"
#include "USB Switch Timer.h"
#include "LegalDlg.h"

#include "QuestionnaireDlg.h"

extern int legal_agree;


// CLegalDlg dialog

IMPLEMENT_DYNCREATE(CLegalDlg, CDHtmlDialog)

CLegalDlg::CLegalDlg(CWnd* pParent /*=NULL*/)
	: CDHtmlDialog(CLegalDlg::IDD, CLegalDlg::IDH, pParent)
{
}

CLegalDlg::~CLegalDlg()
{
}

void CLegalDlg::DoDataExchange(CDataExchange* pDX)
{
	CDHtmlDialog::DoDataExchange(pDX);
}

BOOL CLegalDlg::OnInitDialog()
{
	CDHtmlDialog::OnInitDialog();
	return TRUE;  // return TRUE  unless you set the focus to a control
}

BEGIN_MESSAGE_MAP(CLegalDlg, CDHtmlDialog)
	ON_BN_CLICKED(IDLEGALOK, OnBnClickedLegalok)
	ON_BN_CLICKED(IDLEGALCANCEL, OnBnClickedLegalcancel)
	ON_BN_CLICKED(IDC_LEGALAGREE, OnBnClickedLegalagree)
	ON_EN_CHANGE(IDC_LEGAL, OnEnChangeLegal)
END_MESSAGE_MAP()

BEGIN_DHTML_EVENT_MAP(CLegalDlg)
	DHTML_EVENT_ONCLICK(_T("ButtonOK"), OnButtonOK)
```

```
        DHTML_EVENT_ONCLICK(_T("ButtonCancel"), OnButtonCancel)
END_DHTML_EVENT_MAP()



// CLegalDlg message handlers

HRESULT CLegalDlg::OnButtonOK(IHTMLElement* /*pElement*/)
{
        OnOK();
        return S_OK;  // return TRUE  unless you set the focus to a control
}

HRESULT CLegalDlg::OnButtonCancel(IHTMLElement* /*pElement*/)
{
        OnCancel();
        return S_OK;  // return TRUE  unless you set the focus to a control
}


// if the continue button is pressed check to see if agreement box is checked then
continue
// else prompt for the user to check the box

void CLegalDlg::OnBnClickedLegalok()
{
        // TODO: Add your control notification handler code here
        if (legal_agree == 1)
                {
                CQuestionnaireDlg Dlg2;
                OnCancel();
                Dlg2.DoModal();
        }
        else
        {
                MessageBox("You must agree to continue.","Legal Agreement");
        }
}

// dismiss the dialog if quit button is pressed
void CLegalDlg::OnBnClickedLegalcancel()
{
        // TODO: Add your control notification handler code here
        OnCancel();
}

// when the check button is clicked, change value
void CLegalDlg::OnBnClickedLegalagree()
```

```
{
	// TODO: Add your control notification handler code here
	if (legal_agree == 0)
	{
		legal_agree = 1;
	}
	else
	{
		legal_agree = 0;
	}
}

void CLegalDlg::OnEnChangeLegal()
{
	// TODO:  If this is a RICHEDIT control, the control will not
	// send this notification unless you override the CDHtmlDialog::OnInitDialog()
	// function and call CRichEditCtrl().SetEventMask()
	// with the ENM_CHANGE flag ORed into the mask.

	// TODO:  Add your control notification handler code here
}
```

VII. QuestionnaireDlg.h

```
// QuestionnaireDlg.h

#pragma once

// CQuestionnaireDlg dialog

class CQuestionnaireDlg : public CDHtmlDialog
{
	DECLARE_DYNCREATE(CQuestionnaireDlg)

public:
	CQuestionnaireDlg(CWnd* pParent = NULL);   // standard constructor
	virtual ~CQuestionnaireDlg();

// Overrides
	HRESULT OnButtonOK(IHTMLElement *pElement);
	HRESULT OnButtonCancel(IHTMLElement *pElement);

// Dialog Data
	enum { IDD = IDD_QUESTIONNAIRE, IDH = IDR_HTML_QUESTIONNAIREDLG };

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	virtual BOOL OnInitDialog();

	DECLARE_MESSAGE_MAP()
	DECLARE_DHTML_EVENT_MAP()
public:
	afx_msg void OnBnClickedAge1();
	afx_msg void OnBnClickedAge2();
	afx_msg void OnBnClickedAge3();
	afx_msg void OnBnClickedAge4();
	afx_msg void OnBnClickedAge5();
	afx_msg void OnBnClickedAge6();
	afx_msg void OnBnClickedAge7();
	afx_msg void OnBnClickedCompusage1();
	afx_msg void OnBnClickedCompusage2();
	afx_msg void OnBnClickedCompusage3();
	afx_msg void OnBnClickedCompusage4();
	afx_msg void OnBnClickedCompusage5();
	afx_msg void OnBnClickedCompusage6();
	afx_msg void OnBnClickedOk();
	afx_msg void OnBnClickedCancel();
};
```

VIII. QuestionnaireDlg.cpp

```
// QuestionnaireDlg.cpp : implementation file
// This is the file which controls the questionnaire dialog

#include "stdafx.h"
#include "USB Switch Timer.h"
#include "QuestionnaireDlg.h"
#include "StartTestDlg.h"

extern int user_age;
extern int user_computer_usage;

// CQuestionnaireDlg dialog

IMPLEMENT_DYNCREATE(CQuestionnaireDlg, CDHtmlDialog)

CQuestionnaireDlg::CQuestionnaireDlg(CWnd* pParent /*=NULL*/)
	: CDHtmlDialog(CQuestionnaireDlg::IDD, CQuestionnaireDlg::IDH, pParent)
{
}

CQuestionnaireDlg::~CQuestionnaireDlg()
{
}

void CQuestionnaireDlg::DoDataExchange(CDataExchange* pDX)
{
	CDHtmlDialog::DoDataExchange(pDX);
}

BOOL CQuestionnaireDlg::OnInitDialog()
{
	CDHtmlDialog::OnInitDialog();
	return TRUE;  // return TRUE  unless you set the focus to a control
}

BEGIN_MESSAGE_MAP(CQuestionnaireDlg, CDHtmlDialog)
	ON_BN_CLICKED(IDC_AGE1, OnBnClickedAge1)
	ON_BN_CLICKED(IDC_AGE2, OnBnClickedAge2)
	ON_BN_CLICKED(IDC_AGE3, OnBnClickedAge3)
	ON_BN_CLICKED(IDC_AGE4, OnBnClickedAge4)
	ON_BN_CLICKED(IDC_AGE5, OnBnClickedAge5)
	ON_BN_CLICKED(IDC_AGE6, OnBnClickedAge6)
	ON_BN_CLICKED(IDC_AGE7, OnBnClickedAge7)
	ON_BN_CLICKED(IDC_COMPUSAGE1, OnBnClickedCompusage1)
	ON_BN_CLICKED(IDC_COMPUSAGE2, OnBnClickedCompusage2)
```

```
        ON_BN_CLICKED(IDC_COMPUSAGE3, OnBnClickedCompusage3)
        ON_BN_CLICKED(IDC_COMPUSAGE4, OnBnClickedCompusage4)
        ON_BN_CLICKED(IDC_COMPUSAGE5, OnBnClickedCompusage5)
        ON_BN_CLICKED(IDC_COMPUSAGE6, OnBnClickedCompusage6)
        ON_BN_CLICKED(IDOK, OnBnClickedOk)
        ON_BN_CLICKED(IDCANCEL, OnBnClickedCancel)
END_MESSAGE_MAP()

BEGIN_DHTML_EVENT_MAP(CQuestionnaireDlg)
        DHTML_EVENT_ONCLICK(_T("ButtonOK"), OnButtonOK)
        DHTML_EVENT_ONCLICK(_T("ButtonCancel"), OnButtonCancel)
END_DHTML_EVENT_MAP()



// CQuestionnaireDlg message handlers

HRESULT CQuestionnaireDlg::OnButtonOK(IHTMLElement* /*pElement*/)
{
        OnOK();
        return S_OK;  // return TRUE  unless you set the focus to a control
}

HRESULT CQuestionnaireDlg::OnButtonCancel(IHTMLElement* /*pElement*/)
{
        OnCancel();
        return S_OK;  // return TRUE  unless you set the focus to a control
}


// sets the appropriate value for user_age

void CQuestionnaireDlg::OnBnClickedAge1()
{
        // TODO: Add your control notification handler code here
        user_age = 1;
}

void CQuestionnaireDlg::OnBnClickedAge2()
{
        // TODO: Add your control notification handler code here
        user_age = 2;
}

void CQuestionnaireDlg::OnBnClickedAge3()
{
        // TODO: Add your control notification handler code here
        user_age = 3;
```

```
}

void CQuestionnaireDlg::OnBnClickedAge4()
{
	// TODO: Add your control notification handler code here
	user_age = 4;
}

void CQuestionnaireDlg::OnBnClickedAge5()
{
	// TODO: Add your control notification handler code here
	user_age = 5;
}

void CQuestionnaireDlg::OnBnClickedAge6()
{
	// TODO: Add your control notification handler code here
	user_age = 6;
}

void CQuestionnaireDlg::OnBnClickedAge7()
{
	// TODO: Add your control notification handler code here
	user_age = 7;
}

// sets the appropriate value for user_computer_usage

void CQuestionnaireDlg::OnBnClickedCompusage1()
{
	// TODO: Add your control notification handler code here
	user_computer_usage = 1;
}

void CQuestionnaireDlg::OnBnClickedCompusage2()
{
	// TODO: Add your control notification handler code here
	user_computer_usage = 2;
}

void CQuestionnaireDlg::OnBnClickedCompusage3()
{
	// TODO: Add your control notification handler code here
	user_computer_usage = 3;
}

void CQuestionnaireDlg::OnBnClickedCompusage4()
{
```

```
        // TODO: Add your control notification handler code here
        user_computer_usage = 4;
}

void CQuestionnaireDlg::OnBnClickedCompusage5()
{
        // TODO: Add your control notification handler code here
        user_computer_usage = 5;
}

void CQuestionnaireDlg::OnBnClickedCompusage6()
{
        // TODO: Add your control notification handler code here
        user_computer_usage = 6;
}

// if the continue button is clicked

void CQuestionnaireDlg::OnBnClickedOk()
{
        // TODO: Add your control notification handler code here

        char errors[75] = " ";


        // check to see if both radio boxes are checked
        if (user_age == 0 || user_computer_usage == 0)
        {
                // else write an appropriate error
                if (user_age == 0)
                {
                strcat(errors,"Please select your age group.\n");
                }
                if (user_computer_usage == 0)
                {
                strcat(errors,"Please select your computer usage.\n");
                }
                MessageBox(errors,"Questionaire");
        }
        else
        {

        // dismiss this dialog and call the start test dialog
        OnCancel();
        CStartTestDlg Dlg1;
        Dlg1.DoModal();
        }
}
```

```
// if the quit button is pressed, dismiss the dialog
void CQuestionnaireDlg::OnBnClickedCancel()
{
        // TODO: Add your control notification handler code here
        OnCancel();
}
```

IX. StartTestDlg.h

```
// StartTestDlg.h

#pragma once

// CStartTestDlg dialog

class CStartTestDlg : public CDHtmlDialog
{
	DECLARE_DYNCREATE(CStartTestDlg)

public:
	CStartTestDlg(CWnd* pParent = NULL);   // standard constructor
	virtual ~CStartTestDlg();

	void StartTest(int test, bool openclose, bool redo);
	void OnStart();
	void OnStop();
	void OnClose();
	void ReadAndWriteToDevice();
	void ReadReport();
	void WriteReport();
	void Next();
	void CollectErrors();
	void EndTest();


// Overrides
	HRESULT OnButtonOK(IHTMLElement *pElement);
	HRESULT OnButtonCancel(IHTMLElement *pElement);

// Dialog Data
	enum { IDD = IDD_DIALOG1, IDH = IDR_HTML_STARTTESTDLG };

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	virtual BOOL OnInitDialog();
	bool FindTheHID();
	void DisplayInputReport();
	void DisplayReceivedData(char ReceivedByte);
	void GetDeviceCapabilities();
	void PrepareForOverlappedTransfer();
	void ScrollToBottomOfListBox(USHORT idx);
	void OnTimer(UINT nIDEvent);

	DECLARE_MESSAGE_MAP()
```

```
        DECLARE_DHTML_EVENT_MAP()
};
```

X. StartTestDlg.cpp

```
// StartTestDlg.cpp : implementation file
// This is the main test procedure dialog, it calls all the prompt dialogs.
// it also controls the order of the test and output.

#include "stdafx.h"
#include "USB Switch Timer.h"
#include "USB Switch TimerDlg.h"
#include "StartTestDlg.h"
#include "Resource.h"

#include "ClickSingleDlg.h"
#include "ClickDoubleDlg.h"
#include "ClickTripleDlg.h"
#include "ClickLongDlg.h"
#include "ClickSingleRedo.h"
#include "ClickDoubleRedo.h"
#include "ClickTripleRedo.h"
#include "ClickLongRedo.h"



#include <fstream.h>

extern "C" {
#include "hidsdi.h"
#include <setupapi.h>
}

extern void WINAPI HIDIOCompletionRoutine(DWORD, DWORD, LPOVERLAPPED);

//Application global variables
        DWORD                   Actual;
        DWORD                   BytesRead;
        HIDP_CAPS               Capabilities;
        DWORD                   cbBytesRead;
        PSP_DEVICE_INTERFACE_DETAIL_DATA     detailData;
        bool                    DeviceDetected;
        HANDLE                  DeviceHandle;
        DWORD                   dwError;
        HANDLE                  hEventObject;
        HANDLE                  hDevInfo;
        GUID                    HidGuid;
        OVERLAPPED              HIDOverlapped;
        char                    InputReport[4];
        ULONG                   Length;
```

```
	LPOVERLAPPED	lpOverLap;
	DWORD			NumberOfBytesRead;
	HANDLE			ReadHandle;
	ULONG			Required;
	CHAR			path_file_name[11];
	WORD			path_file_size;
	bool			running = FALSE;
	CString			Buffer[20];	//contains buffered data from the switch
	int			Current_Buffer_Size=0; //current size of the buffer
	int			Previous_Buffer_Size; //previous size of the buffer - used
for picking up incomplete actions
	int			Counter=0;		//increments if buffer stays the same
on subsequent timer events
	int			No_Reports=0;		//the number of reports expected from
each action
	CString			Errors;			//errors during the experiment are
placed here
	CString			Results;		//experimental result string to be
written to the file
	int			test_position=0;	//current test position
	UINT_PTR		Timer;			//timer!
	CDialog			*m_pDlg;		//pointer to a dialog - used for
opening and closing modeless dialogs
	Int			test_sequence[40] = {	3,4,1,1,3,1,2,1,3,4,
							2,2,3,2,1,2,1,3,3,2,
							3,4,4,1,4,1,2,3,4,2,
							2,2,3,4,1,4,3,4,1,4};
	CString			test_results[40][5];	//Array into which the results are
added after buffering
	extern CFile	out_file;			//The output.csv file used for the
results
	extern CString test_headings[100];
	extern int		user_age;
	extern int		user_computer_usage;
	extern int		unfinished;
	bool			Redo=FALSE;
	extern int		halt;

// CStartTestDlg dialog

IMPLEMENT_DYNCREATE(CStartTestDlg, CDHtmlDialog)

CStartTestDlg::CStartTestDlg(CWnd* pParent /*=NULL*/)
	: CDHtmlDialog(CStartTestDlg::IDD, CStartTestDlg::IDH, pParent)
{
	//Contains all actions performed when Start Test Dialog Box is opened

	//Detect Device
```

```
	FindTheHID();

	// Initialise the Buffer
	int i;

	for (i=0;i<20;i++)
	{Buffer[i]="000000";}
	Current_Buffer_Size=0;
	Previous_Buffer_Size=0;

	//Initialise the test results array
	int j;

	for (i=0;i<40;i++)
	{
		for (j=0;j<5;j++)
		{
			test_results[i][j] = "000000";
		}
	}
}

CStartTestDlg::~CStartTestDlg()
{
}

void CStartTestDlg::DoDataExchange(CDataExchange* pDX)
{
	CDHtmlDialog::DoDataExchange(pDX);
}

BOOL CStartTestDlg::OnInitDialog()
{
	CDHtmlDialog::OnInitDialog();
	return TRUE;  // return TRUE  unless you set the focus to a control
}

BEGIN_MESSAGE_MAP(CStartTestDlg, CDHtmlDialog)
	//{{AFX_MSG_MAP(CStartTestDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_WM_CLOSE()
	ON_WM_TIMER()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_DHTML_EVENT_MAP(CStartTestDlg)
```

```
        DHTML_EVENT_ONCLICK(_T("ButtonOK"), OnButtonOK)
        DHTML_EVENT_ONCLICK(_T("ButtonCancel"), OnButtonCancel)
END_DHTML_EVENT_MAP()



// CStartTestDlg message handlers

HRESULT CStartTestDlg::OnButtonOK(IHTMLElement* /*pElement*/)
{
        Results = "";
        Errors = "";
        StartTest(test_sequence[test_position], TRUE, FALSE);
        return S_OK;  // return TRUE  unless you set the focus to a control
}

HRESULT CStartTestDlg::OnButtonCancel(IHTMLElement* /*pElement*/)
{
        if (unfinished == 1)
        {
                //Write the results and errors of unfinished test to a file

                //Questionnaire

                char temp[2];

                _itoa(user_age,temp,10);
                Results += temp;
                Results += ",";
                _itoa(user_computer_usage,temp,10);
                Results += temp;
                Results += ",";

                // Experiment

                int i;
                int j;

                for(i=0;i<40;i++)
                {
                        for(j=0;j<5;j++)
                        {
                        if (test_results[i][j] != "000000")
                                {
                                Results += test_results[i][j];
                                Results += ",";
                                }
                        }
```

```
                        if   ((test_results[i][0]   ==   "000000")&&(test_results[i][1]   ==
"000000")&&(test_results[i][2]         ==         "000000")&&(test_results[i][3]         ==
"000000")&&(test_results[i][4] == "000000"))
                        {
                                Results += ",";
                        }
                }
                char newtemp[3];
                _itoa(halt,newtemp,10);

                // append the halt to the error file

                Errors += "E:Test Halted ";
                Errors += newtemp;

                // append the errors to the results
                Results += Errors;
                Results += "\n";

                //write the results to the file
                out_file.Write(Results, Results.GetLength() );
        }

        OnStop();
        OnClose();
        OnCancel();
        return S_OK;  // return TRUE  unless you set the focus to a control
}


// This function handles all the calling and dismissing of the prompts for different
clicks

void CStartTestDlg::StartTest(int test, bool openclose, bool redo)
{

        //Detect whether the dialog needs to be opened or closed
        if (openclose)
        {

        //Start Timer
        OnStart();

        //Detect whether the dialog needs to be a retry one
        if (redo)
        {
                Errors += "R:";
                char temp[3];
```

```
        _itoa(test_position,temp,10);
        Errors += temp;
        Errors += ".";

switch (test)
{
        case 1:
                        No_Reports = 2;

                        // Display the modal redo a Single Click dialog box
                        m_pDlg = new CClickSingleRedo;
                        m_pDlg->Create(IDD_CLK_SINGLE_REDO);
                        m_pDlg->ShowWindow(SW_SHOW);

                        break;

        case 2:
                        No_Reports = 4;

                        // Display the modal redo a Double Click dialog box
                        m_pDlg = new CClickDoubleRedo;
                        m_pDlg->Create(IDD_CLK_DOUBLE_REDO);
                        m_pDlg->ShowWindow(SW_SHOW);

                        break;

        case 3:
                        No_Reports = 6;

                        // Display the modal redo a Triple Click dialog box
                        m_pDlg = new CClickTripleRedo;
                        m_pDlg->Create(IDD_CLK_TRIPLE_REDO);
                        m_pDlg->ShowWindow(SW_SHOW);

                        break;

        case 4:
                        No_Reports = 2;

                        // Display the modal redo a Long Click dialog box
                        m_pDlg = new CClickLongRedo;
                        m_pDlg->Create(IDD_CLK_LONG_REDO);
                        m_pDlg->ShowWindow(SW_SHOW);

                        break;
}
}
else
```

```
{
switch (test)
{
	case 1:
			No_Reports = 2;

			// Display the modal Single Click dialog box
			m_pDlg = new CClickSingleDlg;
			m_pDlg->Create(IDD_CLK_SINGLE);
			m_pDlg->ShowWindow(SW_SHOW);

			break;

	case 2:
			No_Reports = 4;

			// Display the modal Double Click dialog box
			m_pDlg = new CClickDoubleDlg;
			m_pDlg->Create(IDD_CLK_DOUBLE);
			m_pDlg->ShowWindow(SW_SHOW);

			break;

	case 3:
			No_Reports = 6;

			// Display the modal Triple Click dialog box
			m_pDlg = new CClickTripleDlg;
			m_pDlg->Create(IDD_CLK_TRIPLE);
			m_pDlg->ShowWindow(SW_SHOW);

			break;

	case 4:
			No_Reports = 2;

			// Display the modal Long Click dialog box
			m_pDlg = new CClickLongDlg;
			m_pDlg->Create(IDD_CLK_LONG);
			m_pDlg->ShowWindow(SW_SHOW);

			break;
}
}
}
else
{
switch (test)
```

```
        {
                // close the dialog box

                case 1:
                                m_pDlg->EndDialog(IDCANCEL);
                                delete m_pDlg;
                                break;

                case 2:
                                m_pDlg->EndDialog(IDCANCEL);
                                delete m_pDlg;
                                break;

                case 3:
                                m_pDlg->EndDialog(IDCANCEL);
                                delete m_pDlg;
                                break;

                case 4:
                                m_pDlg->EndDialog(IDCANCEL);
                                delete m_pDlg;
                                break;
        }

        //Stop Timer
        OnStop();
        }
}

// routines butchered from uswitch

void CStartTestDlg::OnStart()
{
        if (running == FALSE)
        {
                //Enable periodic exchanges of reports.
                //Start by reading and writing one pair of reports.
                ReadAndWriteToDevice();

                running = TRUE;

                //Enable the timer to cause periodic exchange of reports.
                //The second parameter is the number of milliseconds between report
requests.
                Timer = SetTimer(ID_CLOCK_TIMER, 50, NULL);
        }
}
```

```
void CStartTestDlg::OnStop()
{
	if (running)
	{
		//Disable the timer.
		KillTimer(Timer);
		running = FALSE;
	}
}

void CStartTestDlg::OnClose()
{
	//Anything that needs to occur on closing the application goes here.
	//Free any resources used by previous API calls and still allocated.

	//Close open handles.
	CloseHandle(DeviceHandle);
	CloseHandle(ReadHandle);
}

void CStartTestDlg::OnTimer(UINT nIDEvent)
{
	//The timer event.
	//Read and Write one pair of reports.
	ReadAndWriteToDevice();

	// if there are enough reports for the current click
	if (Current_Buffer_Size >= No_Reports)
	{
		// dismiss current dialog
		StartTest(test_sequence[test_position], FALSE, Redo);

		// if the test is not finished yet
		if (test_position < 40)
		{
			// call the next function
			Next();
		}
		// else end the test
		else
		{
			EndTest();
		}
	}

	// the routine calls the redo dialog after a set amount of cycles
	if (Current_Buffer_Size > 0)
	{
```

```
                // if the buffer size hasn't changed since the last cycle
                // (i.e. the user has made no further presses)
                if (Current_Buffer_Size == Previous_Buffer_Size)
                {
                        Counter++;
                        // when counter reaches cycle limit (60 = 3 seconds approx)call the
redo dialog
                        if (Counter > 60)
                        {
                                Redo=TRUE;
                                StartTest(test_sequence[test_position], FALSE, TRUE);
                                Current_Buffer_Size = 0;
                                Counter = 0;
                                StartTest(test_sequence[test_position], TRUE, TRUE);
                        }
                }
                else
                {
                        // reset the counter
                        Previous_Buffer_Size = Current_Buffer_Size;
                        Counter = 0;
                }
        }
}


// the dialog which calls the next dialog and prepares the output
void CStartTestDlg::Next()
{
        int i;

        for (i=1;i<No_Reports;i++)
        {
                // chuck away the first value
                test_results[test_position][i-1] = Buffer[i];
        }
        // collect errors if extra clicks are detected
        if (Current_Buffer_Size > No_Reports)
        {
                CollectErrors();
        }
        // reset values
        No_Reports = 0;
        Current_Buffer_Size = 0;
        Previous_Buffer_Size = 0;
        test_position++;
        Redo = FALSE;
        // call the next dialog
```

```
        StartTest(test_sequence[test_position], TRUE, FALSE);

}

// collect the extra outputs
void CStartTestDlg::CollectErrors()
{
        Errors += "E";
        Errors += ":";
        int i;
        for (i=No_Reports;i<=Current_Buffer_Size; i++)
        {
                Errors += test_headings[test_position + i];
                Errors += Buffer[i];
                Errors += ".";
        }
}

void CStartTestDlg::EndTest()
{
        //Write the results and errors to a file

        //Questionnaire

        char temp[2];

        _itoa(user_age,temp,10);
        Results += temp;
        Results += ",";
        _itoa(user_computer_usage,temp,10);
        Results += temp;
        Results += ",";

        // Experiment

        int i;
        int j;

        for(i=0;i<40;i++)
        {
                for(j=0;j<5;j++)
                {
                if (test_results[i][j] != "000000")
                        {
                        Results += test_results[i][j];
                        Results += ",";
                        }
                }
```

```
        }
        Results += Errors;
        Results += "\n";
        out_file.Write(Results, Results.GetLength() );

        OnStop();
        OnCancel();
        OnClose();
        test_position = 0;
        // display the thankyou box
        MessageBox("Thankyou for your co-operation","Test Completed");
}

// These functions are more or less unchanged except where noted with GB:

bool CStartTestDlg::FindTheHID()
{
        //Use a series of API calls to find a HID with a matching Vendor and Product ID.

        HIDD_ATTRIBUTES                                         Attributes;
        SP_DEVICE_INTERFACE_DATA                devInfoData;
        bool                                                    LastDevice = FALSE;
        int                                                             MemberIndex   =
0;
        bool                                                    MyDeviceDetected     =
FALSE;
        LONG                                                    Result;

        //These are the vendor and product IDs to look for.
        //Uses Lakeview Research's Vendor ID.
        const unsigned int VendorID = 0x0925;
        const unsigned int ProductID = 0x1234;

        Length = 0;
        detailData = NULL;
        DeviceHandle=NULL;

        /*
        API function: HidD_GetHidGuid
        Get the GUID for all system HIDs.
        Returns: the GUID in HidGuid.
        */

        HidD_GetHidGuid(&HidGuid);

        /*
        API function: SetupDiGetClassDevs
        Returns: a handle to a device information set for all installed devices.
```

```
        Requires: the GUID returned by GetHidGuid.
        */

        hDevInfo=SetupDiGetClassDevs
                (&HidGuid,
                NULL,
                NULL,
                DIGCF_PRESENT|DIGCF_INTERFACEDEVICE);

        devInfoData.cbSize = sizeof(devInfoData);

        //Step through the available devices looking for the one we want.
        //Quit on detecting the desired device or checking all available devices without
success.
        MemberIndex = 0;
        LastDevice = FALSE;

        do
        {
                MyDeviceDetected=FALSE;

                /*
                API function: SetupDiEnumDeviceInterfaces
                On return, MyDeviceInterfaceData contains the handle to a
                SP_DEVICE_INTERFACE_DATA structure for a detected device.
                Requires:
                The DeviceInfoSet returned in SetupDiGetClassDevs.
                The HidGuid returned in GetHidGuid.
                An index to specify a device.
                */

                Result=SetupDiEnumDeviceInterfaces
                        (hDevInfo,
                        0,
                        &HidGuid,
                        MemberIndex,
                        &devInfoData);

                if (Result != 0)
                {
                        //A device has been detected, so get more information about it.

                        /*
                        API function: SetupDiGetDeviceInterfaceDetail
                        Returns: an SP_DEVICE_INTERFACE_DETAIL_DATA structure
                        containing information about a device.
                        To retrieve the information, call this function twice.
                        The first time returns the size of the structure in Length.
```

```
                        The second time returns a pointer to the data in DeviceInfoSet.
                        Requires:
                        A DeviceInfoSet returned by SetupDiGetClassDevs
                        The     SP_DEVICE_INTERFACE_DATA     structure     returned     by
SetupDiEnumDeviceInterfaces.

                        The final parameter is an optional pointer to an SP_DEV_INFO_DATA
structure.
                        This application doesn't retrieve or use the structure.

                        If retrieving the structure, set
                        MyDeviceInfoData.cbSize = length of MyDeviceInfoData.
                        and pass the structure's address.
                        */

                        //Get the Length value.
                        //The call will return with a "buffer too small" error which can be
ignored.
                        Result = SetupDiGetDeviceInterfaceDetail
                                (hDevInfo,
                                &devInfoData,
                                NULL,
                                0,
                                &Length,
                                NULL);

                        //Allocate memory for the hDevInfo structure, using the returned
Length.
                        detailData = (PSP_DEVICE_INTERFACE_DETAIL_DATA)malloc(Length);

                        //Set cbSize in the detailData structure.
                        detailData -> cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

                        //Call the function again, this time passing it the returned buffer
size.
                        Result = SetupDiGetDeviceInterfaceDetail
                                (hDevInfo,
                                &devInfoData,
                                detailData,
                                Length,
                                &Required,
                                NULL);

                        //Open a handle to the device.

                        /*
                        API function: CreateFile
                        Returns: a handle that enables reading and writing to the device.
```

```
Requires:
The DevicePath in the detailData structure
returned by SetupDiGetDeviceInterfaceDetail.
*/

DeviceHandle=CreateFile
        (detailData->DevicePath,
        GENERIC_READ|GENERIC_WRITE,
        FILE_SHARE_READ|FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        0,
        NULL);

TRACE("CreateFile\n");

/*
API function: HidD_GetAttributes
Requests information from the device.
Requires: the handle returned by CreateFile.
Returns: a HIDD_ATTRIBUTES structure containing
the Vendor ID, Product ID, and Product Version Number.
Use this information to decide if the detected device is
the one we're looking for.
*/

//Set the Size to the number of bytes in the structure.
Attributes.Size = sizeof(Attributes);

Result = HidD_GetAttributes
        (DeviceHandle,
        &Attributes);

TRACE("HidD_GetAttributes\n");

//Is it the desired device?
MyDeviceDetected = FALSE;


if (Attributes.VendorID == VendorID)
{
        if (Attributes.ProductID == ProductID)
        {
                //Both the Product and Vendor IDs match.
                MyDeviceDetected = TRUE;
                //TRACE("Device detected\n");
                //Get the device's capablities.
                GetDeviceCapabilities();
```

```
                                PrepareForOverlappedTransfer();
                        } //if (Attributes.ProductID == ProductID)

                        else
                                //The Product ID doesn't match.
                                CloseHandle(DeviceHandle);
                } //if (Attributes.VendorID == VendorID)

                else
                        //The Vendor ID doesn't match.
                        CloseHandle(DeviceHandle);

        //Free the memory used by the detailData structure (no longer needed).
        free(detailData);
        }  //if (Result != 0)

        else
                //SetupDiEnumDeviceInterfaces returned 0, so there are no more
devices to check.
                LastDevice=TRUE;

        //If we haven't found the device yet, and haven't tried every available
device,
        //try the next one.
        MemberIndex = MemberIndex + 1;

    } //do
    while ((LastDevice == FALSE) && (MyDeviceDetected == FALSE));

    if (MyDeviceDetected == FALSE)
            TRACE("Device not detected\n");
    else
            TRACE("Device detected");

    //Free the memory reserved for hDevInfo by SetupDiClassDevs.
    SetupDiDestroyDeviceInfoList(hDevInfo);

    return MyDeviceDetected;
}

void CStartTestDlg::GetDeviceCapabilities()
{
    //Get the Capabilities structure for the device.
    PHIDP_PREPARSED_DATA   PreparsedData;

    /*
    API function: HidD_GetPreparsedData
```

```
        Returns: a pointer to a buffer containing the information about the device's
capabilities.
        Requires: A handle returned by CreateFile.
        There's no need to access the buffer directly,
        but HidP_GetCaps and other API functions require a pointer to the buffer.
        */

        HidD_GetPreparsedData
                (DeviceHandle,
                &PreparsedData);

        /*
        API function: HidP_GetCaps
        Learn the device's capabilities.
        For standard devices such as joysticks, you can find out the specific
        capabilities of the device.
        For a custom device, the software will probably know what the device is capable
of,
        and the call only verifies the information.
        Requires: the pointer to the buffer returned by HidD_GetPreparsedData.
        Returns: a Capabilities structure containing the information.
        */

        HidP_GetCaps
                (PreparsedData,
                &Capabilities);


        //No need for PreparsedData any more, so free the memory it's using.
        HidD_FreePreparsedData(PreparsedData);
}

void CStartTestDlg::PrepareForOverlappedTransfer()
{
        //Get another handle to the device for the overlapped ReadFiles.
        ReadHandle=CreateFile
                (detailData->DevicePath,
                GENERIC_READ|GENERIC_WRITE,
                FILE_SHARE_READ|FILE_SHARE_WRITE,
                NULL,
                OPEN_EXISTING,
                FILE_FLAG_OVERLAPPED,
                NULL);

        //Get an event object for the overlapped structure.

        /*API function: CreateEvent
        Requires:
```

```
          Security attributes or Null
          Manual reset (true). Use ResetEvent to set the event object's state to non-
signaled.
          Initial state (true = signaled)
          Event object name (optional)
        Returns: a handle to the event object
        */

        if (hEventObject == 0)
        {
                hEventObject = CreateEvent
                        (NULL,
                        TRUE,
                        TRUE,
                        "");

        //Set the members of the overlapped structure.
        HIDOverlapped.hEvent = hEventObject;
        HIDOverlapped.Offset = 0;
        HIDOverlapped.OffsetHigh = 0;
        }
}

void CStartTestDlg::ReadAndWriteToDevice()
{
        //If we haven't done so already, find the device and learn its capabilities.
        //Then send a report and request a report.
        //The test device firmware (usbhidio) adds 1 to each byte received in an OUT
report
        //and sends the results back in the next IN report.

        //Clear the List Box (optional).

        {

        //If the device hasn't been detected already, look for it.
        if (DeviceDetected==FALSE)
                DeviceDetected=FindTheHID();

        if (DeviceDetected==TRUE)
                {
                        //TRACE("Device detected true \n");
                        //Write a report to the device.
                        WriteReport();
                        //Read a report from the device.
                        ReadReport();
                }
        }
```

```
}

void CStartTestDlg::ReadReport()
{
        CString ByteToDisplay = "";
        CString MessageToDisplay = "";
        CString strBytesRead = "";
        DWORD   Result;

        //Read a report from the device.

        /*API call:ReadFile
        'Returns: the report in InputReport.
        'Requires: a device handle returned by CreateFile
        '(for overlapped I/O, CreateFile must be called with FILE_FLAG_OVERLAPPED),
        'the Input report length in bytes returned by HidP_GetCaps,
        'and an overlapped structure whose hEvent member is set to an event object.
        */

        Result = ReadFile
                (ReadHandle,
                InputReport,
                Capabilities.InputReportByteLength,
                &NumberOfBytesRead,
                (LPOVERLAPPED) &HIDOverlapped);




        /*API call:WaitForSingleObject
        'Used with overlapped ReadFile.
        'Returns when ReadFile has received the requested amount of data or on timeout.
        'Requires an event object created with CreateEvent
        'and a timeout value in milliseconds.
        */

        Result = WaitForSingleObject
                (hEventObject,
                6000);
        //DisplayLastError("WaitForSingleObject: ") ;

        switch (Result)
        {
        case WAIT_OBJECT_0:
                {
                //TRACE("wait object \n");
                //DisplayData(ValueToDisplay);
                 strBytesRead.Format("%s%d", "No. Bytes Read: ",NumberOfBytesRead );
```

```
                break;
                }
        case WAIT_TIMEOUT:
                {
                //ValueToDisplay.Format("%s", "ReadFile timeout");
                //DisplayData(ValueToDisplay);
                //Cancel the Read operation.

                /*API call: CancelIo
                Cancels the ReadFile
         Requires the device handle.
         Returns non-zero on success.
                */

                Result = CancelIo(ReadHandle);

                //A timeout may mean that the device has been removed.
                //Close the device handles and set DeviceDetected = False
                //so the next access attempt will search for the device.
                CloseHandle(ReadHandle);
                CloseHandle(DeviceHandle);
                TRACE("Can't read from device\n");
                DeviceDetected = FALSE;
                break;
        default:
                //ValueToDisplay.Format("%s", "Undefined error");
                break;
                }
        }

        /*
        API call: ResetEvent
        Sets the event object to non-signaled.
        Requires a handle to the event object.
        Returns non-zero on success.
        */

        ResetEvent(hEventObject);

        //Display the report data.
        DisplayInputReport();
}

void CStartTestDlg::WriteReport()
{
        //Send a report to the device.

        //The maximum size of an output report. (This can be increased).
```

```
        const unsigned short int      MAXREPORTSIZE = 256;

        DWORD   BytesWritten = 0;
        INT             Index =0;
        CHAR    OutputReport[MAXREPORTSIZE];
        ULONG   Result;
        CString strBytesWritten = "";

        //The first byte is the report number.
        OutputReport[0]=0;

        //Can set the other report values here, or get them from the combo boxes.
        OutputReport[1]=33;
        OutputReport[2]=6;
        OutputReport[3]=15;


        /*
        API Function: WriteFile
        Sends a report to the device.
        Returns: success or failure.
        Requires:
        The device handle returned by CreateFile.
        The Output Report length returned by HidP_GetCaps,
        A report to send.
        */

        Result = WriteFile
                (DeviceHandle,
                OutputReport,
                Capabilities.OutputReportByteLength,
                &BytesWritten,
                NULL);

        if (Result == 0)
        {
                //The WriteFile failed, so close the handle, display a message,
                //and set DeviceDetected to FALSE so the next attempt will look for the
device.
                CloseHandle(DeviceHandle);
                CloseHandle(ReadHandle);
                DeviceDetected = FALSE;
        }

        //Display the result of the API call and the report bytes.
        strBytesWritten.Format("%s%d", "Bytes Written: ", BytesWritten);
        }
```

```
/*
Display-related routines
*/

void CStartTestDlg::DisplayInputReport()
{
        USHORT  ByteNumber;
        CHAR    ReceivedByte,x,y;
        int total,string_index;
        CString timing_string("000000");
        CString comma(",");
        total = 0;
               for   (ByteNumber=0;   ByteNumber   <   Capabilities.InputReportByteLength;
ByteNumber++)
        total += InputReport[ByteNumber];
               if(total != 0)
               {
                       //Display the received data in the Bytes Received List boxes.
                       //Step through the received bytes and display each.
                       //for             (ByteNumber=0;             ByteNumber             <
Capabilities.InputReportByteLength; ByteNumber++)
                       TRACE("\n");
                       string_index = 0;
                       if(running){
                               for   (ByteNumber   =   Capabilities.InputReportByteLength   -
1;ByteNumber > 0;  ByteNumber--)
                               {
                               //Get a byte.
                               ReceivedByte = x = InputReport[ByteNumber];
                               //Get high nibble
                               y = x & 0xF0;
                               //move to low nibble
                               y  = y >> 4;
                               y &= 0xF;
                               (y < 10) ? TRACE("%c",(y + '0')) : TRACE("%c", (y + 'A' -
10));
                               (y < 10) ?  y += '0' : y = y + 'A' - 10;
                               //out_file.Write(&x,1);
                               timing_string.SetAt(string_index,y);
                               string_index++;
                               //Get low nibble
                               x &= 0xF;
                               (x < 10) ? TRACE("%c",(x + '0')) : TRACE("%c", (x + 'A' -
10));
                               (x < 10) ? x += '0': x = x + 'A' - 10;
                               timing_string.SetAt(string_index,x);
                               string_index++;
                               }
```

```
                        //GB: Added Buffering for Error Catching.
                        //out_file.Write(timing_string,6);
                        Buffer[Current_Buffer_Size] = timing_string;
                        Current_Buffer_Size++;
                    }
                }
}
```

XI. ClickSingleDlg.h

```
// ClickSingleDlg.h
// it is almost identical to the other headers of the
// dialogs for double, triple and long clicks, as well as the prompts
// for redone clicks

#pragma once

// CClickSingleDlg dialog

class CClickSingleDlg : public CDHtmlDialog
{
        DECLARE_DYNCREATE(CClickSingleDlg)

public:
        CClickSingleDlg(CWnd* pParent = NULL);   // standard constructor
        virtual ~CClickSingleDlg();
// Overrides
        HRESULT OnButtonOK(IHTMLElement *pElement);
        HRESULT OnButtonCancel(IHTMLElement *pElement);

// Dialog Data
        enum { IDD = IDD_CLK_SINGLE, IDH = IDR_HTML_CLICKSINGLEDLG };

protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual BOOL OnInitDialog();

        DECLARE_MESSAGE_MAP()
        DECLARE_DHTML_EVENT_MAP()
};
```

XII. ClickSingleDlg.cpp

```
// ClickSingleDlg.cpp : implementation file
// This is the file which controls the single click dialog
// it is almost identical to the other dialogs for double, triple
// and long clicks, as well as the prompts for redone clicks

#include "stdafx.h"
#include "USB Switch Timer.h"
#include "ClickSingleDlg.h"

extern int             unfinished;
extern int             test_position;
extern int             halt;

// CClickSingleDlg dialog

IMPLEMENT_DYNCREATE(CClickSingleDlg, CDHtmlDialog)

CClickSingleDlg::CClickSingleDlg(CWnd* pParent /*=NULL*/)
	: CDHtmlDialog(CClickSingleDlg::IDD, CClickSingleDlg::IDH, pParent)
{
}

CClickSingleDlg::~CClickSingleDlg()
{
}

void CClickSingleDlg::DoDataExchange(CDataExchange* pDX)
{
	CDHtmlDialog::DoDataExchange(pDX);
}

BOOL CClickSingleDlg::OnInitDialog()
{
	CDHtmlDialog::OnInitDialog();
	return TRUE;  // return TRUE  unless you set the focus to a control
}

BEGIN_MESSAGE_MAP(CClickSingleDlg, CDHtmlDialog)
END_MESSAGE_MAP()

BEGIN_DHTML_EVENT_MAP(CClickSingleDlg)
	DHTML_EVENT_ONCLICK(_T("ButtonOK"), OnButtonOK)
	DHTML_EVENT_ONCLICK(_T("ButtonCancel"), OnButtonCancel)
END_DHTML_EVENT_MAP()



// CClickSingleDlg message handlers

HRESULT CClickSingleDlg::OnButtonOK(IHTMLElement* /*pElement*/)
{
	OnOK();
	return S_OK;  // return TRUE  unless you set the focus to a control
```

```
}


// if the quit button is pressed, dismiss the dialog and indicate in the halt position
// and unfinished variables

HRESULT CClickSingleDlg::OnButtonCancel(IHTMLElement* /*pElement*/)
{
        unfinished = 1;
        halt = test_position;
        OnCancel();
        return TRUE;  // return TRUE  unless you set the focus to a control
}
```