# Performance Improvement and Feature Enhancement of WriteOn

Samantha Chandrasekar

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Science and Applications

Joseph G. Tront, Chair
Calvin J. Ribbens
Osman Balci

May 28, 2008
Blacksburg, Virginia

Keywords: Tablet PC, Electronic Ink, Computer-assisted teaching, Screen Capture Codec

# Performance Improvement and Feature Enhancement of WriteOn

Samantha Chandrasekar

## ABSTRACT

A Tablet PC is a portable computing device which combines a regular notebook computer with a digitizing screen that interacts with a complementary electronic pen stylus. The pen allows the user to input data by writing on or by tapping the screen. Like a regular notebook computer, the user can also perform tasks using the mouse and keyboard. A Tablet PC gives the users all the features of a regular notebook computer along with the support to recognize, process, and store electronic/digital ink, enabling a user to make and save hand-written notes or data. In institutions of teaching and learning, instructors often use computer-based materials like web pages, PowerPoint® slides, etc., to explain subject matter. The ability to annotate on presentation information using the electronic stylus of a Tablet PC has attracted the attention of the academic community to use the Tablet PC as a potential tool for increasing the effectiveness of presentations in teaching and learning. Tablet PC-based applications such as OneNote®, WindowsJournal® and Classroom Presenter have been developed to enhance note-taking in classrooms based on the fact that a pen stylus is a more natural form of input device for making notes on the computer as compared to the regular keyboard and mouse. Although tools like OneNote®, WindowsJournal® enhanced the note-taking process on the Tablet PC, they lacked the ability to allow the user to directly annotate on the lecture content. Classroom Presenter provides the ability to integrate classroom notes and the presentation material by allowing the instructors and students to annotate over the lecture material. However, all the above tools lacked the ability to allow a user to take notes over the output window of an arbitrary application like Excel, an active simulator or a movies players output. The Tablet PC based tool, WriteOn, developed at Virginia Tech, addresses this drawback.

WriteOn, when deployed on the Tablet PC in a classroom environment, allows the instructor to utilize electronic ink to annotate on top of any application window visible on the Tablet PC display screen, including those that play active content like a movie or simulation.

WriteOn facilitates a user to annotate over a dynamic application window by activating its virtual transparency surface called the *eVellum* (electronic vellum). The user can view a movie or an active simulation running in the eVellum background because of its transparent color. The user can deactivate the eVellum to make it invisible by "piercing" it if he/she wishes to access the desktop or an application window under the vellum window. WriteOn provides the instructor with the ability to broadcast a composite of the dynamic lecture content and ink annotations to the students in real-time. The term dynamic lecture contents is meant to indicate that the content being annotated need not be static words on a background, but may also be window contents that are changing in time. Using WriteOn, the students can make their own notes by writing on the eVellum enabled on top of the lecture stream window without losing visibility of the lecture. The instructor/student can save the ink annotations along with base lecture material as a movie file. The ability of WriteOn to improve classroom presentation and student note-taking as shown by initial tests, were pedagogically very useful. However, in order to deploy WriteOn on large scale in classrooms as an active and effective teaching tool of choice, several aspects of the application had to be improved.

One aspect of the application that needed improvement was the user interface. The primitive Graphical User Interface (GUI) of the WriteOn tool was not easily usable by instructors and students from non-computer science backgrounds. The second aspect needing improvement was the operational performance of the application in terms of its CPU resource utilization. The WriteOn tool has shown to have operational performance issues during the screen capture process. This research therefore aims to address improvements in the GUI to make it more user friendly and increase the operational performance to the point where the user does not notice degradation of a base lecture application. Incorporation of these improvements has led us to rename the application as WriteOn1.0.

WriteOn1.0 implements a picture-based GUI that comprises of two forms: a main form that appears shortly after WriteOn1.0 starts and a toolbar. The WriteOn1.0 toolbar appears in the center of the top edge of the display as soon as the user initiates a task like a screen recording session, by clicking on the appropriate menu button on the main form. The toolbar provides the user, accessibility to perform all the desired activities like annotating, screen recording, presentation broadcast, and piercing of the eVellum by a single-click of the appropriate menu icon. Tool tips that appear when the user points the mouse over a picture icon on the toolbar,

explain the task that shall be performed when he/she clicks on the underlying menu icon. WriteOn1.0 introduces a window-like resizable and movable eVellum called the scalable eVellum that it activates in the area of interest specified by the user. Unlike the first implementation of the eVellum which had a fixed location and spanned the entirety of the user's desktop window, the instructor/student define the dimensions of the scalable eVellum and can choose to re-dimension, relocate and pierce through it at any point of time during a session. WriteOn1.0 also introduces the transparent mode of operation wherein the instructor/student, without having to deactivate the scalable eVellum can access any underlying window by a right-click of the mouse on the eVellum surface while the ink annotations are intact on the foreground,.

WriteOn1.0 addresses the operational performance issues observed during a screen capture session in WriteOn by capturing the activities only in the area of interest of the user for recording and broadcasting. By combining this scheme with a with a lossless screen capture codec called the MSU screen capture codec that has a high-compression ratio and that is optimized for speed for data compression, WriteOn1.0 greatly improves the operational CPU performance of the tool.

WriteOn1.0 employs various technologies to implement its features. The improvements to operational performance are implemented by using the MSU screen codec from Moscow State University's Graphics and Media Lab. Microsoft®'s Video for Windows Framework (VfW) and WindowsMedia Player API's are used to realize the module that records the screen activities to an AVI file while DirectShow of DirectX and ConferenceXP API's are used for streaming presentations over a network. WriteOn1.0, with its features like its scalable eVellum, good operational performance and picture-based GUI is aimed at potentially making it a teaching tool of choice across classrooms and changing the method of classroom instruction of courses involving dynamic content.

# Acknowledgements

I would like to first express my deepest gratitude to my advisor, Dr. Joseph G. Tront for giving me this wonderful opportunity to work with him on this very interesting project. Without his endless guidance, support, encouragement and patience, this research effort would not have been possible. I would like to thank Dr.Osman Balci for his valuable advice and comments on my work. I would like to extend a very special thanks to Dr.Ribbens for all his help and advice.

I appreciate all the help extended to me from the Conference XP team at Microsoft and several of the technical newsgroups for patiently listening to my concerns and addressing them. I would like thank my friend, Jon Howarth for his comments and help with the user interface design.

I would like to thank Monica, Ranjit, Anurag, Pardha and Hari for all the cookouts and moral support during my stay in Blacksburg.

Words cannot express my appreciation and gratefulness to my parents, Uma Chandrasekar and R.Chandrasekar for their unconditional love and support all through these years. I could have not been successful in this endeavor if not for them. Special thanks to my sister Charanya Chandrasekar for being my pillar of support and for making me laugh even when I was about to cry.

Finally, I would like to extend my heartfelt appreciation to my husband, Raghavendra Nyamagoudar for being with me all the time. I cannot imagine myself going through this without his affection and support.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Effective classroom instruction can be a difficult and daunting task for teachers across various disciplines, especially for those teaching subjects which have non-text based lecture material like an animated clip or computer-based software. The use of teaching aids such as the traditional blackboard and chalk or an overhead projector and transparency film to illustrate and elaborate subject matter are not very helpful in instructing such subjects because these teaching methods not only fail to capture and preserve the dynamic nature of the lecture content, but also fail to deliver it to the students. The use of computers to present and teach subjects involving active content is a potentially viable solution for engineering classrooms. A prime example of a computer-based application extensively used in classrooms is the Microsoft PowerPoint. Although this software helps the instructor deliver a lecture in a structured fashion by organizing the presentation content comprising of text as well as non-text animation clips into slides, it lacks the ability to allow the instructor to write notes directly over the lecture slides.

The ability to annotate on a lecture presentation using the pen stylus of the Tablet PC has attracted the attention of academics to use them as a potential tool for effective classroom instruction. Tablet PC-based educational software like Classroom Presenter [1] allows the instructor to supplement lecture material comprising of static text and image only PowerPoint slides by directly annotating on them. However, computer science and engineering classes often involve the use of non-text based lecture material like a simulation demo over which a lecturer would like to make annotations. In order to facilitate this, we present a Tablet PC-based educational tool called WriteOn1.0, the second core implementation of the WriteOn [2] tool also developed at Virginia Tech. In this thesis, we describe WriteOn1.0 and the implementation of its many features. We also detail the design of the new version of the tool and how its features can potentially aid in effective lecturing, especially for subject matter like programming and digital circuit design, both of which involve the use of dynamic presentation material.

## 1.1 Overview

In order to effectively teach complex subject matter, good instructors always strive to explain it in a manner easily understood by their students. In an attempt to present subject matter in an interesting way so that the students assimilate concepts with ease, instructors often resort to using various teaching aids. While traditional teaching aids help the instructor to deliver only text-based and image-based subject matter, they lack the ability to allow the instructor to save the lecture and do not allow the student to perform note-taking on the lecture material while it is being delivered. In order to take notes, the student often has to shift his/her attention from the presentation screen/blackboard to his/her notebook. Comprehensive and organized note-taking is an important practice for effective learning. Studies have demonstrated that good classroom note-taking have an increased effect on student cognition [3]. Students taking courses on topics like logic circuits would often be interested in noting down the various steps involved in developing and running a logic circuit simulation which can be difficult to record using the traditional pen and paper paradigm. Moreover, traditional teaching tools lack the ability to present and help instruct non-textual lecture content in classrooms.

Research by Kornefors and Lundberg [4] has proved the importance of the use of computers as an aid for effective teaching. An example of a computer-based application extensively used in classrooms is Microsoft PowerPoint. Although animated PowerPoint slides present a lecture in a well-organized and structured fashion, they do not allow the instructor to illustrate a topic by directly writing over the slides. This deficiency tends to leave the instructor writing on a blackboard to supplement the electronic presentation. The difficulty is that the mixed mode presentation can be confusing to students and less than exacting for the presenter – the annotations do not relate closely enough to the animation. The students have to rely on a text editor like Notepad® to type their notes into or they can make handwritten notes on printed copies of the lecture slides. Such note-taking methods are not very effective because the student has to often focus his/her attention from the lecture in order take notes and shift his attention back to the lecture so that he/she does not lose track of the presentation.

The Tablet PC, with its features that include the ease of input of natural handwriting with a pen-like stylus and its ability to store digital ink in the natural handwritten form, can potentially enhance classroom note-taking. Applications such as OneNote®[5], Journal® [6] and Classroom Presenter take advantage of the Tablet PC's ability to accept and store hand written notes in

digital format in order improve student note-taking. Classroom Presenter is a tablet PC-based educational software which allows instructors and students to annotate on PowerPoint slides. The ability to make notes directly on the lecture content presented in PowerPoint slides is a very powerful capability that can potentially change the way students and instructors make notes in classrooms. Wolfman [7] illustrates in detail, several advantages of using the Classroom Presenter tool in the classroom as a teaching aid. Wilkerson, Griswold and Simon [8] elaborate on the use of Classroom Presenter in a virtual classroom environment. Research done by Tront [9] describes how the Classroom Presenter tool enhances and improves the active learning process in classrooms based on the use of features such as instructor annotations, student note taking on personal copies of lecture slides, and in-class student submissions.

Tablet PC-based tools like Classroom Presenter help the instructor to effectively explain a lecture presentation involving text and image only content by allowing him/her to annotate directly on the presentation information using the stylus in a Tablet PC, However, topics such as programming and simulation have to be taught from both the theoretical and practical point of view. The practical portion of the lecture often involves computer-based non-text content that cannot be easily explained using PowerPoint slides. An instructor can explain a topic in programming or simulation in a better fashion if he/she is able to walk the student through the execution process in and, is able to make annotations on the executing software window to illustrate the subject matter. WriteOn [2] , developed in Virginia Tech facilitates an instructor to effectively teach subjects involving text as well as non-static lecture material by enabling him/her to write on top of any operational software window using the electronic stylus.

WriteOn realizes the ability to apply electronic ink on any open and active window on the Windows desktop area by enabling a virtually transparent entity called the electronic vellum (eVellum). The eVellum is overlaid over almost the entire desktop area. The user can write on the virtual transparency while any application windows under the vellum operate visibly in the visible background. With the implementation of WriteOn, it became possible for instructors to prepare and present lectures on topics involving dynamic computer-based content in real-time and directly annotate on the lecture content using e-ink. WriteOn enables the students to make notes while viewing the presentation on their own local system. By enabling the eVellum on their desktop screen, the students can make notes over the window that renders the live lecture material being broadcast to each individual. WriteOn provides a feature by which students can

record the composite of the live lecture stream, the instructor annotations, and their own e-ink annotations as a movie file on their computer. Initial tests of WriteOn in the classroom have shown that it helps improve the presentation material and promote active-learning [10]. The tool WriteOn forms the base topic of the research effort presented in this thesis. The following section gives an overview of the research problem addressed in this thesis dissertation.

## 1.2 The problem and proposed solution

WriteOn was developed as a Tablet PC-based teaching tool to help improve and enhance lecture presentations, student note-taking, and promote active classroom learning with its features such as the eVellum, recording of the screen activities and e-ink on the eVellum to a movie file and the live broadcast of lecture information along with the instructor annotations. Using WriteOn, students can make notes in handwritten form on top of the static and dynamic lecture material, and save local copies of the same for future reference. Beta tests were conducted to gather initial feedback about the tool in an active classroom. WriteOn received positive feedback from instructors and students with both indicating an improvement in presentation of lecture material and student note-taking [2, 10]. Instructors observed an improvement in student participation and active classroom learning. However, suggestions from initial users of WriteOn made it evident that, in order to deploy WriteOn in sizable classrooms, there was a need to improve some aspects of the system. Necessary improvements cited by the testers include significant improvement to the user interface, a significant increase in efficiency of the screen capture and an absolute necessity in increase of efficiency of the broadcast process.

The user interface needed to be upgraded in order to help instructors from any background to easily perform typical tasks on the system, like initiating a broadcast session, without having to input sometimes unfamiliar parameters, like the bit rate, channel information, port number, etc. The original steps to toggle the eVellum (make a visible eVellum invisible and vice-versa) by accessing the WriteOn tray icon on the taskbar had to be simplified. The efficiency of the screen recording process needed improvement both in terms of the size of the movie file generated as well as the utilization of CPU cycles. The first version of WriteOn was observed to consume all available CPU cycles during a screen capture process. Such consumption of CPU resources is unacceptable especially when you consider that the purpose of a teaching tool like WriteOn is to allow programs like simulations to run in the background while they are visually

being annotated over using e-ink. The lag and occasional loss of data at the student side of broadcasted presentations had to be minimized. Another originally unavailable feature of WriteOn is the ability to record the lecture material on the instructor machine while he/she is broadcasting it to the students.

The research work presented in this thesis addresses the suggested improvements needed in the first implementation of WriteOn, along with some value-added features like, the ability to simultaneously record a lecture and broadcast it, the ability to context switch to an application window below the eVellum without toggling it, etc. The new version of the WriteOn tool developed in this thesis will be referred to as WriteOn1.0.

The remaining portion of this chapter briefly summarizes the main features of WriteOn1.0 namely, a new feature called scalable eVellum, the picture-based GUI, the facility to broadcast and save a presentation in parallel and finally the MSU screen codec for data compression.

### 1.2.1 The scalable eVellum

The development of the virtual transparency, the eVellum in the first version of WriteOn, is an advance in computer-assisted tools to explain subjects that involve computer-based dynamic content like simulations or programming. When the almost transparent eVellum is enabled on the Tablet PC, instructors have the ability to annotate over any application ranging from those displaying textual data to ones that play dynamic content like movies while they operate visibly under the eVellum. Users can access any underlying application window by toggling the eVellum off with a click on the appropriate menu item. This mechanism of deactivating the eVellum at any point in time to access any operating application window is useful while teaching subjects like programming where an instructor often come across “what-if” questions from students. Such questions often call for change in some parameters of a simulation and running it again to see the results. One of the observed issues in the first version of WriteOn is that a mode switch from the eVellum mode to the pierce mode (in which the eVellum is turned off) has to always be performed in order to access any underlying window like the ones displaying lecture material before responding to a “what-if” question posed by a student. This is because the eVellum is enabled almost over the entire desktop area. The compulsory mode switch of the eVellum in order to access the control buttons of an operating software window before addressing a “what-if” question lead to a certain lag in the response from the instructor.

Another drawback of the original implementation of the eVellum is its fixed size and location. The eVellum is located at the top-left corner of the user's desktop and had fixed dimensions corresponding to almost the entire area of the user's screen. The user lacks the ability to resize the original eVellum to the dimensions of his/her choice and moving it to a location where he/she wishes to. The first version does not provide the user enough flexibility and ease of accessing his folders or other documents on his/her desktop while in the note-taking mode owing to the screen area occupied by the eVellum. The user has to explicitly toggle the eVellum off to perform actions on his desktop window. By replacing the static eVellum of WriteOn with a resizable eVellum we could potentially improve the overall user experience of WriteOn.

WriteOn 1.0 solves the problem of single-sized full screen vellum with a window-like eVellum called the scalable eVellum. This new mechanism can be resized to, and moved over a targeted area of interest, thus eliminating the need to toggle it off before reaching to the underlying application or desktop. The resizable and movable nature of the scalable eVellum gives the instructor and students the ability to focus into and out of portions of the presentation material by re-dimensioning and relocating it to selected portions of the lecture content whenever needed. The user defines the dimensions and the location of the scalable eVellum. The new version of the eVellum has distinct red borders around it to help the user identify it on the screen with ease.

WriteOn1.0 broadcasts and records screen activity that occurs only in the area over which the user enables the scalable eVellum. Since the user is usually interested in annotating only over a certain portion of the lecture presentation as opposed to the entire desktop at any given point of time, the scalable eVellum reduces the amount of data that needs to be compressed and subsequently transmitted to the student.

The scalable eVellum introduces a new mode of operation of the virtual transparency called the transparent mode. In this mode, the user can right click on the scalable eVellum to access the underlying application window without having to toggle the eVellum off. This mode of operation preserves the annotations in the foreground of the eVellum and ensures that instructors and students do not deviate from the base context of the presentation while accessing the underlying window.

### 1.2.2 Picture based GUI

After initial user testing of the first version of WriteOn, there were suggestions to improve the usability of the tool in order to enhance the overall user experience. The instructors found it difficult to comprehend the meaning of required parameter fields like “bit rate”, “height”, “width”, “frames per second”, “type of eVellum”, etc. that had to be set before beginning a lecture session. There were nagging delays when the user tried to activate the menu items from the WriteOn tray icon during a lecture broadcast session because the operational performance issues of the screen capture process caused a lag in the response time of the tool. Initial users requested a minimization of the time taken to switch between eVellum mode and the pierce mode. WriteOn1.0 addresses the user interface issues with the implementation of a picture-based GUI.

The new picture-based GUI of WriteOn1.0 aims at reducing the effort required to complete desired tasks on the tool. The WriteOn1.0 GUI consists of a main form and a toolbar. The toolbar is similar to the toolbar of the Remote Desktop application in Windows® [11]. This toolbar is the top-most form on the desktop screen and sits above all the applications windows. The user can perform all desired tasks like, enabling/disabling the eVellum, broadcast/receipt of lecture, recording of lecture and change of pen color and width settings with a single click of the appropriate menu icon on the toolbar. Since the toolbar is thin in width, it does not interfere with the visibility of any other Windows desktop application. The switch between the eVellum mode and pierce mode can be done by a single click of an icon on the toolbar. With the switch between eVellum modes a single click away, the new version of WriteOn can potentially help in significantly promoting active learning and the quintessential what-if questions in a classroom by almost completely eliminating the lag involved in a context switch from the tool to the operating software window. The new GUI eliminates the need to enter the unfamiliar data values like the frame rate, width and height before a screen capture session, helping improve the overall user experience.

In addition to making improvements to already available WriteOn features, WriteOn1.0 implements some valuable new features. An important and noteworthy one is a feature by which an instructor can simultaneously record a local copy of the live lecture stream he/she is broadcasting to a class.

### 1.2.3 Saving live lecture presentation

The ability to broadcast presentation material along with all the e-ink annotations and active and/or static content from the instructor machine allows the students to view the complete lecture in real-time. In many cases, an instructor may want to record a copy of the lecture he/she is presenting in a session to a movie file while he/she is instructing in the classroom. This currently unavailable feature in the first version of WriteOn is implemented in WriteOn1.0. Using WriteOn1.0 the instructor(s) can record in parallel, the live lecture stream along with all the e-ink annotations that he/she is broadcasting over the network to their local machine for later review and enhancement of the course material. The lecturer can make any additional notes or key points on the recorded classroom lecture before he/she uploads it on a file share server for the students to download. This gives the instructors an option to augment the live lecture material before handing out to the students contributing to better lecture notes. The instructor can also choose to reuse the recorded material from one lecture session to present to another set of students who have taken the same subject, eliminating the need to prepare or pre-record the lecture in advance.

In order to be able to perform two parallel screen recording sessions, one for broadcasting and another for saving as a movie file, the operational performance of the capture process must be optimized for speed, vellum size, resolution and memory. Any issues in the performance of the screen capture process like high CPU cycle utilization would cause the tool to stall and lead to unacceptable data-loss. WriteOn1.0 solves the performance issues of the screen capture process of the first version of WriteOn with a combination of the scalable eVellum and the use of the MSU (Moscow State University) screen codec [12] to perform video compression. Sub-section 1.2.4 enlists the reasons for choosing the MSU screen codec in WriteOn1.0.

### 1.2.4 Using MSU screen capture codec

The MSU (Moscow State University) screen codec is the video data compressor used in WriteOn1.0 to compress the video stream before broadcasting over a network and recording it to an AVI file. The MSU screen codec has a high data compression ratio that produces compressed frames that are of a small size for broadcast over the network. The reduction in amount of data transferred during broadcast causes a decrease in lag and data loss at the student side. The high compression ratio of the MSU codec also means that a file that is generated when the

instructor/student records the screen activities as a movie is optimized for disk space. The MSU screen codec is highly efficient when it comes to CPU resource utilization. The use of an efficient codec is very important to teaching subjects like logical circuit simulation which involve the use of CPU intensive simulation software as dynamic lecture material. By improving the efficiency of CPU intensive processes like screen capture and video stream compression, the codec augments the enhancement of overall user experience of using WriteOn1.0.

Chapter 2 of this thesis presents some of the related research on Tablet PC-based computer-assisted teaching and learning tools done by other researchers followed by Chapter 3 that gives an overview of the software technologies used in WriteOn1.0. Chapter 4 presents a summary of the main logical components of WriteOn1.0 while Chapters 5, 6, 7, 8 and 9 explain in detail, each of these logical software components. Appendix-I presents the user's manual that explains to user, how to perform tasks on WriteOn1.0 in a step-by-step manner. Appendix-II contains the code snippet of the that implements the scalable eVellum and Appendix-III gives the code that implements the feature to record screen activities to an AVI file using the VfW framework. Important results and conclusions of this research are presented in Chapter 10. Chapter 10 also elucidates potential future work on WriteOn1.0.

# Chapter 2

# Related Work

In this chapter, we shall examine some of the other educational tools used for classroom lecturing such as Classroom Presenter, DyKnow [13] and Camtasia [14]. We shall also review their impact on classroom teaching and learning. The before mentioned tools are presentation tools in the sense that they allow the instructor to not only create lecture presentations, but also act as a platform for delivering them as well. In addition, they allow him/her the ability to apply ink on the lecture content. We shall also talk about some of the Tablet PC based note-taking applications like Windows Journal® and OneNote®. Finally, this chapter will look at a screen-recording tool called CamStudio™ [15] that can be used to create video lectures for distance learning.

## 2.1 Overview of the Classroom Presenter tool

Classroom Presenter facilitates the instructor to augment PowerPoint-based slides with the e-ink of the Tablet PC and projects them to a class. The integration of e-ink with lecture content gives the instructor the ability to enhance the presentation by illustrating important concepts while teaching. Classroom Presenter provides a broadcast facility through which the students can view the instructor annotations and lecture content in real time.

In order to send and view the lecture material, the students and the instructor connect to each other over a TCP/IP-based network. The instructor's machine serves as a TCP server to which each student machine connects. The instructor can directly import PowerPoint slides to Classroom Presenter at the beginning of a lecture. The lecture slides, along with the embedded e-ink can be exported as to a cp3 file after the user opens them in Classroom Presenter. Once the students connect to the instructor's machine, the lecture slides along with instructor ink annotations appear in the Classroom Presenter application window in real-time. Instructors have

the choice of either allowing the students to independently navigate through the entire deck of slides in the presentation or to restrict their view only to the slide visible on the instructor's presentation system.

The use of Classroom Presenter in a lecture session has received positive responses from the students and instructors. The students indicated that the tool increased their attention to the lecture and improved their understanding of the course material. The instructors liked the ability apply e-ink on the lecture slides [16]. Research efforts of Anderson, R. et al. [17] give valuable insight into the impact of using digital ink for lecturing. Classroom Presenter has shown to promote and facilitate classroom interaction, active learning and collaborative learning. In an active classroom session, the instructor can conduct active learning exercises in the form of spontaneous tests and ask the students to work on them within a given period. The students can then annotate over a lecture slide to respond to a question posed by the instructor and then use the submission facility provided by Classroom Presenter to send their answers back to the instructor. The instructor has to enable the submissions option in order to receive student submissions. These active learning exercises not only help the students to learn better but also give the instructor feedback on how well they have understood a concept. The instructor can broadcast some of the students' answers to the class for a group discussion. Simon, B. et al. [18] explains their experience of using the Classroom Presenter for an undergraduate computer architecture class. With e-ink and the mechanism of creating slides only for the instructor's view, the tool facilitated classroom interactivity and student participation through collaborative design exercises. Beth Simon, Ruth Anderson, Crystal Hoyer and Jonathan Su explain their experience and observations of deploying the Tablet PC-based system to achieve active and collaborative learning in computer sciences courses [18].

However, the Classroom Presenter tool greatly reduces the instructor's flexibility to annotate on lecture materials which involve video as it restricts the ink annotation feature to PowerPoint slides only. Since Classroom Presenter is PowerPoint-based, the instructor has to always prepare his/her lecture slides ahead of time. Instructors also indicated an increase in preparation time when the lecture session involved impromptu exercises and student submissions. Although the tool is not very suitable for teaching subjects like digital circuit simulations, the main feature of Classroom Presenter, which is the ability to directly apply

electronic ink on the presentation material has the potential of replacing the pen paper paradigm for note-taking and instruction in classrooms.

## 2.2 Overview of DyKnow

DyKnow (*www.dyknow.com*) is computer-aided teaching tool that is deployable on pen-based computing devices like Tablet PC's as well as regular laptop computers. The DyKnow system allows the instructor to share e-ink-based handwritten notes and sketches as well the lecture material with students in a classroom setup when they directly connect to his/her machine over a network. Using DyKnow, an instructor can type as well as annotate on the lecture material using e-ink for the students to view immediately. The students can add their own notes to the lecture material by typing or writing using the electronic stylus. DyKnow allows the student to save the lecture notes as an electronic document for later review. The student can replay his/her annotations on a stroke-by-stroke basis to review subject matter like mathematical proofs and execution of an algorithm. DyKnow is a very flexible tool in the sense that allows the teacher to import and teach using a wide array of already existent material like PowerPoint slides, web content, snapshots, etc.

DyKnow, like the Classroom Presenter, facilitates the instructor to conduct interactive classroom sessions through mechanisms of student submissions and feedback. The student submission and mechanism can be employed to conduct on spot quizzes. The feedback mechanism allows the teacher to get valuable student input through a polling mechanism. In order to respond to a poll, the student can click on one of three "traffic light" icons namely red, yellow or green to let the instructor know his/her level of understanding of the ongoing lecture. The red icon indicates poor understanding; the yellow one indicates moderate understanding while the green indicates good understanding. Since, this feedback result is not shared among the rest of the class, a student can feel free to let the instructor know his/her level of comprehension. The instructor can use this valuable feedback to adjust the speed of the lecture. DyKnow also facilitates the students to share information with the rest of the classroom. The instructor can enable a student to present his/her ink annotations to the classroom during a session. Such interaction is very useful while solving an algorithmic problem in class, where the students are expected to come up with an optimized solution. Breque, Dave has used DyKnow for instruction

of Computer Science courses in DePauw University. Instruction using DyKnow has shown to enhance the understanding of course material and improve note-taking among students [19]. Research by Rockman et al. [20] has shown that instruction using DyKnow has an increased effect on homework completion, student engagement and a decrease in classroom absences.

Though DyKnow has note-worthy features, an instructor is restricted to use lecture materials like slides, web pages etc. that have to prepared ahead of the lecture. According to a study conducted by Hrepic, Z. et al. [21] in Fort Hays State University for a Physics class indicated that students preferred the instructor to write down and derive a formula than use pre-prepared slides so that they can follow how a formula evolved in a step-by-step manner. In addition, it is not capable of permitting annotations on any dynamic application window like a live movie stream or simulation.

## 2.3 E-ink based note-taking tools

Note-taking is an essential activity that is beneficial to students for effective learning. The ability of the Tablet PC to accept e-ink as freeform handwriting makes it a potentially viable tool for classroom note taking. In a study to research the use of Tablet PC's for note-taking, Turner, S. et al. [22] show that taking notes in the handwritten form rather than typing on the Tablet PC helps the student improve information recall. Research efforts of Intons-Peterson and Fournier (1986) [3] have proven that note-taking improves information recall. In this sub-section, we examine two commonly used e-ink note-taking tools, the Windows Journal and Microsoft OneNote.

Microsoft Windows Journal® (henceforth "Journal") is commercial note-taking tool for the Windows Tablet PC edition. Windows Journal® facilitates the user to make handwritten notes on an interface very similar to a notebook page. The user can also incorporate images into his/her notes and make annotations on them. He/She can also convert the handwritten notes to text using the in-built handwriting recognition module of the Tablet PC. Journal also facilitates a search feature by which a user can search for keywords in his/her handwritten notes. Journal notes can be shared with all users, including those who do not have a Tablet PC, as they can view the Windows Journal® notes from Microsoft Internet Explorer version 5.0 or later. Although the tool serves as a good note-taking tool for classes that primarily involve diagrams and texts, it does not

support the ability to make notes on dynamic content and facilitate online collaborations among users.

Microsoft's OneNote® is a tool for note-taking built for pen-based computing devices. OneNote® displays a window with ruled notebook page where students can annotate on just as they would on a notebook. The virtual pages of handwritten notes can be saved to the local disk as an OneNote® file and can be shared with an instructor if need be. OneNote® facilitates classroom interaction and collaborative learning with its facility to share the electronic notes with either the instructor or another student by initiating a share session from the tool. Although OneNote® provides great note-taking, note-sharing and note-organizing capabilities, it is not as extensible as Classroom Presenter or DyKnow. OneNote® does not allow annotations on application windows playing dynamic content and does not have the facility to broadcast lectures as live stream.

Although e-ink enabled note-taking tools provide a very good alternative to the pencil and notebook, these do not allow the students to incorporate their notes directly into the lecture content and facilitate classroom instruction for courses involving dynamic content.

## 2.4 Overview of VNC

Virtual Network Computing (*VNC*) (*www.realvnc.com*) [23] is a remote desktop tool that can be used in classroom environments for collaborative learning. VNC essentially broadcasts all the desktop activities from a computer to any other computer(s) that connect to it. Using this tool, the instructor can start a presentation on his/her computer and the students can connect to the instructor's machine to view the presentation. Because VNC broadcasts the screen activities on the server computer to the connecting client computers, VNC can be potentially used to teach subjects involving active content like simulations. Using a projector in the class, the instructor can demonstrate a how-to session of any simulation software on a remote computer to the students using the VNC tool. This setup also facilitates the use of videos and animation clips for the purpose of instruction. The use of videos in a classroom can help the student complex concepts and procedures that cannot be easily explained using text and sketches only. VNC promotes a collaborative learning environment in the classroom because a group of students can interact to solve a problem posed on the instructor machine because all of them can access the lecture material. This is possible because, VNC being a remote desktop tool, any mouse and

keyboard event in the students' machine also generates the same events in the instructor machine. Although such interaction promotes collaboration and active learning, it also possesses some risks. Some problems include the modification of lecture material in an undesirable manner since the student can control what is displayed the instructor machine. VNC is that it lacks support for e-ink which means that the students would have to use other applications like OneNote® or Windows Journal® to annotate and make notes. VNC does not scale well in large classrooms due to network congestion issues since students connect via unicast as opposed to multicast.

In the case of a unicast streaming, each client establishes a separate connection to the server machine. Thus, each client has a reserved set of network resources for receiving the data stream. Each client receives its own copy of the data stream. This means that if there are five clients connecting to a server, there shall be five streams broadcasted, one to each client. Hence, unicast streaming can be categorized as a one-to-one connection. In a multicast streaming scenario, all the clients who are interested in receiving the data stream subscribe to a special multicast address by connecting to it. The server streams the data to this multicast address and all clients connected to it can view the stream on their machines. As we can see, there is only one data stream that is broadcasted from the server irrespective of the number of clients receiving the stream unlike the unicast streaming. This implies that a multicast stream requires and uses the same amount of network resources as that of a single unicast stream. Thus, a multicast stream, unlike a unicast stream preserves network bandwidth and can also be used in low bandwidth local area networks. In the case of VNC which is employs unicast streaming the amount of network bandwidth allocated to a client is indirectly proportional to the total number of connections. This means that, if a large number of students connect to single instructor machine, each student gets a small share of the bandwidth which may not be enough to view the data stream, leading to data congestion and delays while relaying the lecture.

## 2.5 Overview of computer screen capture tools: Camtasia Studio and CamStudio™

Camtasia Studio is commercially available screen recording software developed by TechSmith Corporation. Using the Camtasia suite of software, one can create platform

independent video and audio recordings of any activity displayed or any action performed by the user on the user's Windows desktop screen to a movie file. These activities can range from PowerPoint presentations, demo of a software package, or a tutorial on how to create a C++ project using Visual Studio. Camtasia also provides tools to convert the movie file into a Flash (.swf) video, streaming video or podcasts for easy upload and viewing on the internet. Using Camtasia, it is possible to create a video of a lecture presentation with a picture of the instructor by capturing images from a webcam. Camtasia allows the user to define the area of capture and allows him/her the ability to dynamically change the location of the capture area during a recording session. The ability of Camtasia to capture all video components on a user's screen like mouse cursor movements and all application windows even including those playing animation makes it an excellent tool for preparing tutorials and lectures. Smith, L. and Smith, E. [24] present how Camtasia can be used to enhance online learning by developing tutorials on software packages for the students taking courses that involve the use of a particular new software, and uploading it for their review. Wassgren, C. and Krousgrill, C. [25] propose the use of Camtasia with Tablet PC's for classroom and distance learning. Research efforts of Hartsell, T., & Yuen, S. [26] prove the positive effect of using streaming videos created using Camtasia for online learning and student feedback.

However, the earlier versions of the software failed to capture the e-ink annotations on the eVellum of WriteOn while recording the screen activities. Earlier Camtasia versions also lacked the ability to allow the users to apply e-ink and did not permit the user to change the area of capture. These features were included in its release in March 2008, well after WriteOn introduced them in 2005. Camtasia v5.1 allows the user to apply e-ink over the area he/she is capturing. Although Camtasia can be a very useful tool for teaching because of the above features, it lacks the capability to broadcast a lecture in real-time. The transparency in Camtasia can be enabled over any application for annotation but it does not preserve the dynamicity of the application underneath. This means that the user cannot view the execution of a sorting algorithm like insertion sort when he/she enables the ink overlay in Camtasia.

CamStudio™ is an open source and free screen recording software suite similar to Camtasia Studio. CamStudio™ distributes its proprietary codec called the CamStudio™ codec, a recorder and producer. Using the CamStudio™ software, the user can record the screen activities in an area the user selects and compress it to an AVI file using the codec selected by the user

from the menu of CamStudio™. Users can set the other parameters of the recording session like the quality and rate of screen capture (frames per second). CamStudio™ allows the user to relocate the capture area but lacks the feature that allows the user to resize the capture area dimensions. CamStudio™ is a very good tool available for no cost to prepare lecture videos, tutorials or demos in either AVI or SWF (Shock Wave Flash) format. Research efforts by Subhlok J. et al [27] have demonstrated the positive impact of using a hybrid instructional model of using CamStudio™ in conjunction with the Tablet PC to record computer science lectures to provide students "anytime anywhere" [28] learning capability. CamStudio™ has been used in a medical school in Korea to successfully create video lectures as a self-learning tool for the students. However, CamStudio™ is not electronic ink enabled which means the user cannot enable ink overlays for annotating and the software lacks the capability of streaming the lecture in real-time for broadcast.

The WriteOn tool attempted to incorporate the positive features of the tools mentioned above while at the same time, overcoming their shortcomings to provide an easy to use and viable solution that would enrich classroom presentations. WriteOn allowed the instructor to integrate ink and live dynamic content and broadcast it to the students for them to view, save and make notes. The positive role of using electronic ink and dynamic content to enhance classroom teaching and its scope for changing the way instructors teach certain subjects was the main motivation to address the shortcomings of the first version of WriteOn with the implementation of WriteOn1.0. Chapter 3 gives an overview of the .Net framework over which WriteOn1.0 is built followed by a summary of the technologies used to achieve its features of and finally the structural overview of the application.

# Chapter 3

# System Overview

WriteOn1.0 is designed for the Tablet PC running the Windows operating system. WriteOn1.0 is primarily developed in the language C# of the .Net framework. The screen capture and screen recording modules are implemented in C++. The design of WriteOn1.0 follows the well-known Model View Controller (MVC) paradigm [28]. WriteOn1.0 uses the Conference XP API to realize the facility to broadcast the live lecture stream. Windows Media Player API functions implement the feature that allows a user to save a lecture as a movie file on the local disk. This chapter gives a brief introduction to the technologies used to implement WriteOn1.0 followed by the structural description of the tool.

## 3.1 The .Net Framework

The .Net framework [29] is a programming infrastructure developed by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services. .NET is a general-purpose software development platform. Applications built using the .NET Framework can run on any Windows operating system that has the .NET runtime engine installed, without having to recompile them. The .NET framework supports a wide array of high-level languages like C#, VB.Net, C++, etc. to build applications.

The developer writes code using one or more of the high-level languages in an Integrated Development Environment (IDE) like Visual Studio (VS) [30] to develop an application. The compiler associated with the language compiles the source code of the application into Intermediate Language (IL). The Common Language Runtime (CLR) of the .NET Framework then interprets the intermediate code. The .NET core is a virtual machine that turns intermediate language (IL) into machine code. The CLR enables an application that is once built and compiled

on a machine using the .NET framework to execute on any other machine that has the .NET framework. The .NET framework includes an extensive class library featuring all the functionality one might expect a contemporary development platform to support including: Windows GUI development, database access (ADO.NET), web development (ASP.NET) and web services. Figure 3.1 illustrates the steps involved in the compilation and running of an application on the .Net framework.

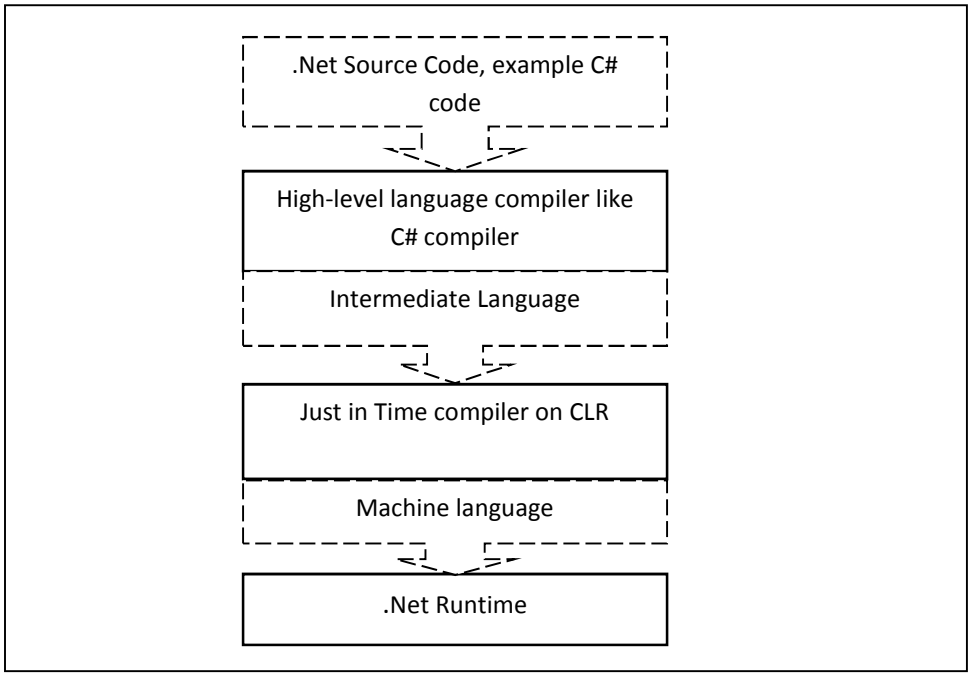



**Figure 3.1: Illustration of compilation and running of .NET project**

In Figure 3.1, the Intermediate Code Compiler first compiles a .NET project developed in C#, Visual Basic and other languages supported in the framework to intermediate code realized by Intermediate Language (IL). The IL is independent of any language constructs which makes it possible to run a .Net application on any Windows operating system which can understand the IL constructs. The Just In Time compiler (JITer) of the CLR converts the intermediate code into machine language. The .Net Runtime runs the application by executing the machine language.

The Microsoft Visual Studio Integrated Development Environment (IDE), widely used to develop .NET applications, generates the intermediate code and produces the executable file in IL. The .Net Runtime executes the IL to run the application. One of main features of the .NET Framework is its cross-language compatibility. This means that .NET components can interact

with each other regardless of the language in which they are developed. An example of cross-language interoperability in WriteOn1.0 is the interoperability between the screen capture module written in C++ and the lecture broadcast module developed in C#. The broadcast module has to receive the stream of screen shots from the screen capture module to broadcast them over the network. The language interoperability and OS independence made .NET the framework of choice to develop WriteOn and WriteOn1.0.

## 3.2 Developing .Net projects

In order to develop and execute any application targeted to the .Net platform, the .NET Framework must be installed. Thus, one of the prerequisites of the Tablet PC on which WriteOn1.0 will be deployed is that the .Net Framework is installed at the correct revision level. The redistributable .NET installation files are available on the Microsoft official website (*http://msdn.microsoft.com/en-us/netframework/default.aspx*) from which it is easily possible to install the .NET framework. Microsoft Visual Studio .NET (version 7) must be installed before we start to develop a new .NET project. The IDE has tutorials and help files which guide the user through the steps to create new projects like C# console applications, new solutions that can be comprised of multiple projects in different languages or a single new file like a class file for a project. Classes and functions that contain their definitions in a .NET dynamic link library (DLL) file can be added to a project. After we put together a project solution file it can be compiled to an executable file by selecting the Build Solution menu item on the main menu of Visual Studio. The executable files can be run by clicking the F5 function key or the run button on the toolbar of the IDE. VS provides options like stack traces, local variable watching using which one can debug an application. The next section of this chapter provides a short overview of the various API's used to develop WriteOn1.0's features.

## 3.3 API used in WriteOn1.0

The following sub-sections provide an overview of the API's used to realize the various features of WriteOn1.0. These include the ConferenceXP API which implements the broadcast module, the Video for Windows (VfW) API [31] used to save a lecture as an AVI file, and the Windows Media Player API used to create a playlist of lecture AVI files.

### 3.3.1 ConferenceXP (CXP) and the ConferenceXP API

The ConferenceXP [32] project is an initiative by the Microsoft® Research team (*http://research.microsoft.com/en-us/projects/conferencexp/default.aspx*) to provide a platform that can empower universities to build real-time collaboration and videoconferencing applications which in turn can be used to enhance the teaching and learning process. The CXP project provides a research platform by providing API's and a framework for designing and implementing distributed and collaborative learning applications. The CXP Software Development Kit (SDK) provides a rich API and infrastructure to build ink-based applications. CXP also makes use of wireless technologies to provide applications over distributed systems. All of the above mentioned capabilities are incorporated into WriteOn1.0 using the CXP SDK and API's. Chapters 8 and 9 describe how CXP API's are used in WriteOn1.0 to achieve the live lecture broadcast ability.

### 3.3.2 Video for Windows (VfW) and AVI file functions

Video for windows (VfW) is a multimedia framework developed by Microsoft (*http://msdn.microsoft.com/en-us/library/ms713492(VS.85).aspx*) to allow users to play digital video files. The VfW technology introduced the AVI (Audio-Video Interleave) file format to store digital video and audio content. The .NET Framework exposes a set of API's which allows the developer to manipulate or add AVI files to their applications. The API's contain the AVIFile functions and macros which are contained in a dynamic link library. These functions are used to build the ScreenCapturetoAVI module, which is used to capture the desktop activities to a compressed video stream and write the compressed data to an AVI file on the local hard disk. Chapter 7 explains the AVIFile functions used in the ScreenCapturetoAVI module.

### 3.3.3 WindowsMedia Player API

The WindowsMedia Player API is a part of the Windows Media Player SDK. It provides the necessary functions to create playlists and modify them. A playlist is a set of media items, like AVI files in our case. Playlists can be created from the Windows Media Player GUI or programmatically. The Windows Media Player object model provides objects and methods to

manipulate both individual media items and playlists. The WMPLib type library exposes the necessary API's required to manage the play lists. Chapter 6 details of how playlists are used in WriteOn1.0.

## 3.4 Structural description of WriteOn1.0

This section gives the structural overview of WriteOn1.0 to describe how the various components of interact. Figure 3.2 contains a block diagram of the major components along with a schematic view of how they interact.

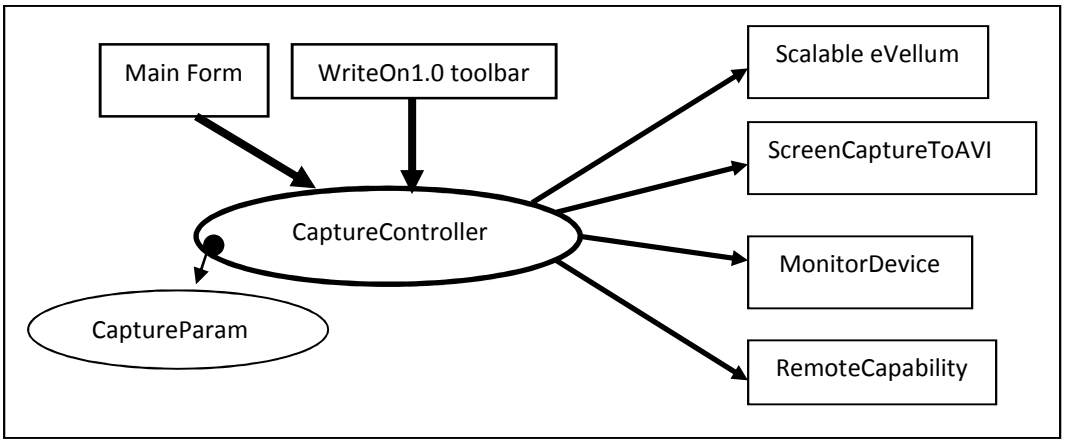



**Figure 3.2 Structural overview of WriteOn1.0**

The new version of WriteOn consists of two forms, the Main form and the Toolbar form. The Main form, as the name suggests, is the main application window of WriteOn1.0 that appears shortly after the user starts the application. The main tasks supported by WriteOn1.0 that include: enabling/disabling eVellum, start/stop lecture broadcast, start/stop receipt of broadcast and screen capture. These operations can all be performed by choosing the appropriate menu button from the main form. WriteOn1.0 produces a tool-bar at the top-center of the top border of the primary display screen once a task is chosen from the main form. This toolbar has icons to perform an array of functions like change of pen color and width settings and take snapshots upon clicking them. The toolbar also has menu items to perform the main tasks supported by WriteOn1.0. One small drawback of WriteOn1.0 is that once it is operative, the area under the toolbar is not usable by any application program since WriteOn will keep its menu display on top of any information being sent to the display by any applications.

The WriteOn1.0 forms, that form the UI component, perform all the user selected tasks by calling functions of the CaptureController class as shown in Figure 3.2. The CaptureController class is the central regulator and controls all the actions to and from all the software components of the application. In order to avoid any state and data inconsistencies of the controller because of all the UI components that access it, only one instance of the controller is shared amongst them. The wrapper class, Controller class, creates a static singleton object of the CaptureController class. All the user defined properties for screen capture, like the dimensions of the capture area, path of the file to which all screen activity must be recorded, etc. are all stored and provided in the form of attribute values of the CaptureParams class object. Since the CaptureController object stores a reference to CaptureParams class object, any changes made to these properties are reflected across all components of the application.

WriteOn1.0 follows the MVC paradigm which is a common architecture used to develop GUI based applications. As seen from Figure 3.2, the Main form and tool bar form along with the other UI components like the eVellum, Monitor Device and RemoteCapability constitute the "View". The CaptureController class forms the "Controller" and the data stored in the CaptureParams form the "Model" of the MVC architecture. The OOP (Object Oriented Programming) paradigm [33] is used to build all the software components in WriteOn1.0.

The next chapter introduces the features and functionality implemented by each of the logical components of WriteOn1.0. The chapter also briefly explains the choice of the technologies and API used to achieve the functionalities. Chapter 6 to Chapter 9 elaborates on the technical details of each of the logical components.

# Chapter 4

# WriteOn 1.0 Components

This chapter presents a brief and general description of the functionality of each of the logical components in WriteOn1.0 and the technology used to achieve them. WriteOn1.0 has four major logical components namely:

- Scalable eVellum
- ScreenCapturetoAVI
- Monitor Device
- Remote Capability

These logical components each implement a feature of WriteOn1.0. Two or more logical components may interact with each other allowing a user to perform a task using WriteOn1.0. Three logical components, namely, scalable eVellum, monitor device and remote capability are implemented in C#. The ScreenCapturetoAVI component is implemented in C++. This chapter presents the highlights of the logical components as well as their impact on the user experience.

## 4.1 Scalable eVellum

Conceptually setup as a virtual transparency, the eVellum can be enabled over any application window running on the Tablet PC. The virtual transparency supports the addition of electronic ink using the pen stylus of the Tablet PC. The eVellum preserves the dynamicity and visibility of the application running in any window beneath it. The original implementation of the eVellum was static in the sense that it, unlike any other window on the user's desktop, it could not be resized or relocated. The rigid size of the eVellum (typically the size of the Tablet PC screen), and its fixed location at the top-left corner of the screen, denies the user any kind of flexibility while using WriteOn because he/she has to always toggle it before accessing the desktop to perform common actions like opening a text file or access another applications. WriteOn1.0 improves the user experience by transforming the eVellum from a single-sized static

transparent borderless form to a resizable and agile one with a distinct border. The new eVellum is called scalable eVellum in WriteOn1.0 because of the ability to move and resize it. Chapter 6 explains the technicalities of the scalable eVellum in detail. The ScreenCapturetoAVI (Section 4.2) module captures the screen activity including standard program action, mouse movements, and e-ink on the scalable eVellum to a movie file.

All the recording of the desktop activities to the local hard disk during a lecture session are saved as a playlist. A playlist is collection of AVI files in WriteOn1.0. While using the scalable eVellum, a user may change its dimensions or location thereby changing the area of screen capture. A new AVI file is created each time the user resizes the scalable eVellum in order to capture activities in its new dimensions. This is because we cannot write frames of different sizes to a single AVI file. Thus, when the capture area is changed, while a recording session is in progress, we have to terminate the current recording session and create a new AVI file. All of the AVI files generated in a single lecture session are grouped together as a single playlist. The Windows Media Player API provides functions to build and manipulate movie playlists. The Windows Media Player API comes as part of the Windows Media Player SDK. The user chooses the name of the playlist before the start of a recording session. Upon the end of a lecture recording, an instance of the WindowsMediaPlayer class is created which exposes the "CreatePlayList" and "AppendItem" functions. The "CreatePlayList" function takes in the name of the playlist as a parameter and creates an empty playlist with that name. The AVI file created is then appended to this playlist by calling the "AppendItem" function with the name of the AVI file as the parameter.

## 4.2 ScreenCapturetoAVI

The ScreenCapturetoAVI module allows the user to record the screen activities on his/her local disk as an AVI file using the Video for Windows (VfW) framework. Originally, the screen recording functionality was implemented in the Monitor Device module in WriteOn. Since the streaming and recording of the screen activities were both implemented in a single module, the user had the choice to either stream the file or save it, but cannot do both simultaneously. The lack of choice to simultaneously save and stream the lecture meant that the instructor could not save a local copy of the live lecture presentation in order review before making it available to the students. To help improve the presentation material, it was necessary that the instructor be able

to stream and save the lecture at the same time. Thus, the function of saving the lecture to a movie file(s) has been moved to separate module in WriteOn1.0 called the ScreenCapturetoAVI. Chapter 7 gives the implementation details of this module.

The Video for Windows (VfW) framework provides all the AVI file functions and macros that are needed to create and manipulate AVI files. These functions are used in the ScreenCapturetoAVI module to record the screen activities to an AVI file on the local hard disk. The user can choose to create an AVI file at any location of his/her choice. In order to optimize space while creating movie files, we need to compress the incoming video stream before writing it to a file. The VfW framework exposes functions which can create a compressed data stream from an uncompressed stream and a compressor like the MSU screen codec.

The MSU Screen Codec is the compressor used to compress the video content captured before it is broadcast or recorded to an AVI file in WriteOn1.0. The codec has a high compression ratio, producing lesser data to broadcast and smaller movie files. The MSU Screen Capture codec is efficient in terms of the use of CPU cycles and its operational efficiency is one of the reasons why the instructor can simultaneously both record and broadcast a lecture presentation. While the ScreenCapturetoAVI saves screen activities to a movie file, the Monitor Device module streams the video over a network Section 4.3 briefly summarizes the functionality of the module Monitor Device.

## 4.3 Monitor Device

The Monitor Device module captures the screen activity on the instructor machine and broadcasts it live to the students. The MSU Lossless Screen Capture codec v1.2 is used to compress the data before streaming the video over the network to the student(s). The Monitor Device module captures only the screen activities and e-ink in the area enclosed by the scalable eVellum for broadcast. As soon as the instructor resizes the scalable eVellum, the monitor device detects it and updates the dimensions of the area of broadcast. WriteOn1.0 uses ConferenceXP API to create the compressed video stream and then broadcast it over the network. Chapter 8 details the working of this component. The Remote Capability module plays the video stream on the student machine as explained in Section 4.4.

## 4.4 Remote Capability

The Remote Capability module decompresses the compressed lecture stream from the instructor and renders it on the student machine in a separate window. In the first version of WriteOn, there was an eVellum enabled by default in the window which rendered the live lecture stream on the student machine. With WriteOn 1.0, static eVellum on the student lecture window is disabled. The student can now enable the new scalable eVellum and save the live lecture to a movie file by choosing the appropriate menu item from the WriteOn1.0 toolbar. Chapter 9 details the working of this component. The Remote Capability implements its functionality using ConferenceXP API.

The logical components of WriteOn1.0 facilitate the user to perform various tasks using the tool. In order to be able to deploy WriteOn1.0 on a large scale, it is important to ensure good overall user experience of the tool. The user experience of the first implementation of the tool was affected in by poor operational performance because of CPU intensive tasks like screen recording. Thus, it became important to address the performance issues of WriteOn as a first step towards improving the user experience. Chapter 5 focuses on the performance issues of the first implementation of WriteOn with regard to the CPU usage and the motivation behind implementing WriteOn1.0. The chapter also explains the technical details of the MSU screen codec as well as the reasons behind choosing this codec in WriteOn1.0.

# Chapter 5

# Solving Performance Issues of WriteOn

The motivation behind the research efforts to enhance and improve WriteOn was the initial positive feedback received from the pilot run using the tool in a classroom setting. Initial user feedback suggested an urgent need to address the operational CPU performance of the tool in order to make WriteOn a tool of choice for instruction in classrooms. The original implementation of the tool consumed almost all available CPU cycles while performing a screen capture operation which was unacceptable. In this chapter, we present the performance issues observed in WriteOn in terms of its CPU usage, the reason we need to address these performance problems, and the solution we implemented in WriteOn1.0.

CPU usage is a metric used to measure the performance of an application software with respect to the amount of CPU resources it uses. The metric represents the processor usage, expressed as a percentage of total capacity. Since the total available CPU cycles are fixed and shared among all the running processes, each process must optimize its CPU resource usage as much as possible in order to ensure the smooth running of all the processes on the system. The CPU usage is a preliminary indicator of the amount of resources a process consumes while it executes. It is inversely proportional to the performance of the application that triggered the process. This means that if a process has a high CPU usage while it runs, it can hinder the overall user experience by causing the whole system to stall, thus preventing the user from performing trivial tasks like opening a folder or file. A single process with a high CPU usage metric can, not only deteriorate the performance of the associated application, but also degrade the performance of all the other active applications as well. Thus, it is always desirable to have a process consume the minimal amount of CPU resources while it operates.

In order to solve the operation CPU performance issues of WriteOn, we need to first profile the application to determine the logical component responsible for the high CPU usage. In software engineering, profiling refers to a performance analysis of any target application in order

to determine which portions of the application needs to be optimized for speed and/or memory. The information needed for the investigation of the program like the memory usage and the program speed are usually gathered while the application executes. Section 5.1 of this chapter explains how the performance of the first version of WriteOn was profiled using the Windows Task Manager. Section 5.2 explains the importance of addressing the issue and Section 5.3 that presents the solution to the performance issues.

## 5.1 Profiling the performance of WriteOn

Tools called *profilers* analyze the performance of a target application. Profilers measure the behavior of the application based on the data gathered while the application executes. WriteOn was profiled using the Windows Task Manager contained in Microsoft Windows. Windows Task Manager is a performance analysis tool included with all Microsoft Windows family of operating systems. It provides information about the CPU and memory usage of all active applications and active processes in addition to network activity statistics. The Task Manager essentially gives the user a summarized version of performance metrics such as CPU and memory usage. The Task Manager can also be used to set process priorities, processor affinity in the case of multiple processors, forcibly terminate processes, and shut down, restart, hibernate or log off from Windows.

The Task Manager can be started by holding down the Ctrl+Alt+Del key combination and the selecting the “Task Manager” button from the menu options on the Windows Security dialogue box. Alternatively, the user can start it by using the Ctrl+Shift+Esc key combination. The Task Manager window consists of four tabs namely, Applications, Processes, Performance and Networking. As the name implies, the applications tab gives the current information on all the active applications running on the system and each of their statuses. The processes tab gives the details of all the processes running on the system. These details include the CPU time, memory usage, name of the process and the name of the user who initiated it. The performance tab gives the graphs of the current system information related to the usage of the CPU and memory resources.

Using the CPU usage and memory graphs from the performance tab as well as the CPU time information from the processes tab, we can probe the operational performance of the WriteOn tool. The CPU Usage graph or meter shows the total current processor usage as a percentage of

its maximum capacity. The CPU Usage History graph gives the plot of the CPU usage data against time. The CPU time can be defined or expressed as percentage of elapsed time that the process used the processor (CPU) to execute instructions.

To begin the performance analysis of WriteOn, we first need to ensure that the performance statistics only reflect the operational CPU and memory usage statistics of the tool only. We set up the system to run only default processes of the WindowsXP Tablet PC and WriteOn. This means that there were no browser windows or any other applications like simulations, music players running. The wallpaper was set to solid black. The CPU usage was measured from the CPU usage meter every ten seconds over a period of five minutes. The average CPU usage was noted at an average of 4 percent of the total CPU capacity. The CPU time of WriteOn.exe was zero all the while.

The next step was to initiate the broadcast process in WriteOn to note its performance during a typical classroom session. The user connected to the virtual classroom at venue 234.4.4.4 in the broadcast/instructor mode.

The screen capture parameters were set as follows:

Frame capture rate: 1 frame/sec

Bit Rate: 1000000

Capture dimensions: 1024 X 768 (The first version of WriteOn captured the activities on the entire screen)

Compressor: WMVideo 9 encoder (The default compressor used in the first version of WriteOn)

The CPU usage and CPU time statistics were noted again for a period of five minutes at a frequency of 10 seconds while broadcasting. The CPU usage was observed at an average of 60%. This 56 percent CPU resource consumption was from the capture and broadcast of the desktop screen and occasional mouse movements on it. However, in a classroom setting, the instructor would like to capture active content like a live simulation for broadcast to the students.

In order to determine the CPU resource consumption in such a scenario, the Logic Works® application was started. Logic Works® is a digital circuit simulation software widely used in classrooms. A sample simulation was initiated to note the performance of WriteOn while capturing and broadcasting active content. While the simulation was running, we observed that the Logic works IDE itself consumes around 65% of the CPU resources. The WriteOn user then connected to a virtual classroom venue 234.4.4.4 and started broadcasting.

It was noted that when the simulator and broadcast process were running simultaneously, WriteOn became somewhat unresponsive when the user tried to perform essential tasks like enabling the eVellum and writing on it. The WriteOn tray menu could not be enabled immediately as well because of the CPU resource constraint. In the first version of the WriteOn tool, the user always had to access the WriteOn tray menu to perform all tasks on WriteOn except disabling and enabling the eVellum. The slow response of the tray menu meant that the user could not perform tasks on WriteOn without any hindrance, therefore degrading the overall user experience. On the simulator side, the simulation was observed to be slower and it stalled periodically during the session. The combined CPU usage was peaking at close to 98% all the time. The CPU time of WriteOn peaked at 95 most of time, thereby hindering the execution of the simulation. This kind of performance statistics is unacceptable for practical teaching purposes.

In addition to the above performance analysis, WriteOn was also profiled for its operational performance during a file record session. In this mode of operation, the user starts a screen capture session and records the screen activities to an AVI file. The performance statistics were observed when WriteOn captured the simulation to a movie file on his/her computer. The CPU usage was noted to be at 70% when recording to a lecture a file. The file size was noted as well. It was noticed that the size of file generated was somewhat large when we used the WindowsMedia 9 codec as is the case with the first version of WriteOn. To compare the difference in the sizes of the AVI files, we generated a movie file by recording the desktop screen for 30 seconds at 1 frame per second. The WindowsMedia 9 codec produced a file of 2.57 MB. On the other hand, the file produced using the MSU Screen capture codec was only 1.3 MB. The difference in the file sizes implies that the amount of data generated for broadcast was also large and was overloading the network leading to the observed loss of packets at the student end.

Since the CPU resources were constrained only when we were in a screen capture mode, we concluded that the screen capture process, in particular the task of compressing data was the most CPU intensive module in the application and that the performance bottle neck was caused by the current codec. A possible solution to the performance issue would be to replace the codec with a codec that is fast and efficient. Section 5.2 explains the implications of a poor performing capture module and the need to address the issue.

## 5.2 Importance of addressing performance issue

The capability to capture, integrate and broadcast dynamic lecture material and e-ink as a single video stream from the instructor machine to the student machine, in real-time is one of the most compelling features of the WriteOn tool. Using this WriteOn feature, an instructor can instruct courses like programming and digital circuits from both the theoretical as well as the practical aspect. By not restricting the instructor to using only text and images for teaching, WriteOn gives the instructor the ability to use any kind of computer-based material for lecturing. This feature makes teaching courses involving active lecture material a simpler task. This is because the instructor need not struggle to make PowerPoint presentations of the important snapshots of a running program or a simulation to illustrate the subject. He/She can just walk the students through a sample execution of the code sample while illustrating it at the same time.

One of the primary requisites of educational software is to permit the smooth execution of an application that forms the base lecture material alongside it. In the case of the WriteOn tool, it implies that in order to be able to effectively teach the simulation, one would need the screen broadcast process as well as the simulation to execute without creating a CPU resource crisis. Since simulation software is usually CPU intensive in nature and we have little control over improving their efficiency, we have to ensure that we optimize the operational performance of our capture and broadcast process as possible. The lack of thereof can stall an ongoing simulation and interfere with the pace of the lecture.

The first version of WriteOn was profiled in a classroom on a Toshiba Protégé Tablet PC that had a Mobile Intel® Pentium III –M, 1.33 GHz processor and 496 MB RAM. One of the consequences of the poor operational CPU performance was the loss of lecture data packets at the student side. Clearly, any loss of data while teaching is not acceptable and does not contribute to effective lecturing and learning. Considering the impact of using a CPU intensive application like WriteOn in a classroom, improving the performance issues arising during the screen capture process became an imperative task. The next section shall present some possible solutions to improving the performance of WriteOn.

## 5.3 Solution to WriteOn tool's performance issues

Optimized operational CPU performance of educational software like WriteOn is a priority since it is often used with base lecture material that itself, is inherently CPU intensive in nature. In the case of the first version of WriteOn, one of the major causes of performance related issues is the lack of a CPU with high processing power. Hence, an easy solution to the performance problem of the tool is to upgrade the hardware to a Tablet which has a dual core processor, for example, the Fujitsu Lifebook T425 Series that have 1.83 GHz Intel® Core Duo Processor and 0.99GB RAM. However, it is not a practical solution since it is not possible for many students to buy new high end Tablet PC's. In order to be able to use WriteOn in all classrooms, including those using Tablet PC's with a single processor with mediocre processing power, it is important that we resolve the performance issues of WriteOn.

Based on the performance analysis of the tool it was observed that the performance issues in the application were due to the WindowsMedia 9 codec that was used for video data compression. Data compression is in itself a CPU intensive task especially when the data involved comprises of a video stream. Using a data compressor that is optimized for CPU resource usage and memory is the key to improving the performance of WriteOn. In addition to its speed, the codec must also produce as small a compressed stream as possible so that we reduce the amount of data broadcasted on the network.

We profiled some non-propriety codecs and compared them with the current codec of WriteOn under various performance metrics. These metrics are explained below.

Compression Ratio: The compression ratio can be technically defined as ratio of the factor by which the size of an uncompressed file is reduced to that of the original uncompressed file. This ratio is an essential metric used in data compression. For example, suppose an uncompressed media file is of 15 MB. Using a suitable codec, the same media file is compressed to a file size of 5 MB. In this specific case, the data compression codec reduces the size of the data file by a factor of three. The compression ratio achieved by the codec thus, would be 3:1.

Speed: This metric specifies the amount of time or resources that the codec uses to compress the data. A codec with a high compression and decompression speed is always desirable.

Section 5.1 presented the sizes of the files generated using various codecs. As evident from the results, the MSU screen capture codec produces files that are an order smaller in magnitude

as compared to the WindowsMedia 9 codec. The MSU codec was noted to have better worst-case operational performance compared to the WindowsMedia 9 codec. The worst-case scenario is when WriteOn records the screen activities of the entire user desktop. Chapter 10 presents the results of the performance analysis. The smaller file size and its performance statistics makes MSU screen capture codec the codec of choice in WriteOn1.0. Section 5.4 presents the features and technical details of the MSU Screen Capture codec. The next section presents an overview of codecs.

## 5.4 MSU lossless screen capture codec

By definition, a codec is a software module that is used to compress or decompress a digital media file, such as a song or video. A compressed file takes up less storage space on your computer and, can be transferred across the internet more quickly and smoothly as compared to an uncompressed file. Codecs are used to create and play nearly all music or video files on a computer or on web sites.

A codec can consist of two components—an encoder and a decoder. The encoder compresses a file during creation, and the decoder decompresses the file so that it can be played or rendered on the screen. There are two forms of data compression or encoding namely, "lossless" and "lossy". As the name suggests, lossless data compression is used when the file has to be uncompressed to produce exactly the same data as was before compression. Text files, for example are stored using lossless techniques, since losing a single character can make the text misleading. However, since there is no room for loss of data, there are limits to the minimum file size that can be obtained with lossless compression. Lossless compression ratios are generally in the range of 2:1 to 8:1. Lossy compression, in contrast, works on the assumption that all the data doesn't have to be compressed to reproduce file of acceptable quality after decompression. Generally, in the lossy compression scheme, much information is simply discarded away from images, video data, and audio data. Because of the reduction in amount of base data, using lossy compression can give compression ratios that can be an order of magnitude greater than those available from lossless methods.

The MSU screen capture is a lossless video compression codec developed by Moscow State University Graphics & Media Lab [13] for lossless compression of video taken from computer

screen. The MSU codec produces highly compressed files due to motion prediction and encoding only the deltas between frames.

The method of encoding in the MSU screen codec is based on the fact that the desktop screen does not change much during a course of time. For example, once a user starts an application or task like opening a word file, he/she would remain at the task for a while. This means that the screen shots at any two intervals of time would be almost identical most of the time. It would thus, suffice to only encode the differences between the frames after the first frame is captured. This method of encoding is called the “delta encoding” or “delta compression” method.

In the delta compression method, the differences between frames are recorded in discrete files called "deltas" or "diffs". Because the differences in the frames are often small, delta encoding greatly reduces data redundancy. From a logical point of view, the deltas or diffs is the information that can be used to produce one frame from the other. In video compression, delta frames can considerably reduce frame size, and are used in virtually every video compression codec. In delta encoded transmission over a network, only a single copy of the original frame is available at each end of the communication channel. Special error control codes are used to detect which parts of the frame has changed since its previous version in order to produce the next frame for rendering. The nature of the data to be encoded influences the effectiveness of a particular compression algorithm. Delta encoding performs best when data has small or constant variation. For an unsorted data set, there may be little to no compression possible with this method.

Using a motion prediction technique, a video codec can predict pixel values from sets of reference pixels usually in other frames. Using a motion vector that formulates how the current frame can be reconstructed based on a reference frame, the codec can create the current frame and render it. Since the codec is capable of reconstructing a frame, it is sufficient to only transmit the motion vector data, thereby increasing the compression efficiency greatly.

Using the technique of delta encoding and motion prediction, the MSU Screen codec produces a high compression ratio and high speed of compression while efficiently using processor resources for encoding screen data, making it the most desirable codec to be used in WriteOn1.0. Because of the MSU codec’s performance statistics, it was possible to realize the feature that permits an instructor to record and transmit the live lecture data in a session in

parallel that would have been impossible to achieve using WindowsMedia 9 video codec. The new version of WriteOn combines the efficiency of the MSU codec with the scalable eVellum to optimize the operational CPU performance of the tool.

Chapter 6 presents the details of the scalable eVellum, another highlighted feature of WriteOn1.0 that is the key to improving user experience when using WriteOn1.0. Combining this new mechanism with the MSU codec, we can greatly improve the performance of WriteOn1.0 by capturing the screen activities only in the area defined of the scalable eVellum.

# Chapter 6

# Scalable Electronic Vellum

The implementation of the electronic vellum otherwise called the eVellum in WriteOn was the result of the research efforts to realize the "annotations anywhere" feature. Undoubtedly, the feature to "ink anywhere" on top of any visible window on the user desktop is one of the most striking capabilities of WriteOn. In technical terms, the eVellum is a transparent borderless windows form having the same dimensions as that of the user desktop screen. The eVellum is the top-most window of all visible windows on the desktop. Since the eVellum is almost transparent, the user can see any application running under the eVellum window and make ink annotations on it using the Tablet PC pen. The eVellum preserves the dynamic nature of all application windows under it. This means that, for example, a user can view a movie playing in a Windows Media Player window at the background while he/she is writing over it using the electronic stylus.

In classrooms, it is common for lecture content to be a combination of static texts and active computer based content like simulations, visual clips, etc. An instructor often switches between text and visual clips to explain the topic from both a theoretical and practical points of view. In order to preserve the activated visible state of the eVellum along with the ink annotations and notes when the instructor switches context, it is setup as the top-most window of all the open windows and forms on the user's Windows desktop. Setting the eVellum as the top-most from also prevents its accidental deactivation, thereby making it invisible when the user switches between windows manually or by using the Alt + Tab key combination.

The early version of WriteOn implemented the screen capture functionality using two technologies, namely DirectX and Windows Media Encoder SDK. In order to capture active content and ink while using either technology, two types of eVellum namely, the "Solid Vellum and the "Overlay Vellum" were implemented. After initial tests, results showed that DirectX technology performed better than the Windows Media Encoder technology (Section 8.5) and is the technology used in WriteOn1.0.

Section 6.1 explains the behavior of an application window on the Windows OS and the need to implement the “overlay vellum” for DirectX and the “solid vellum” for the Windows Media Encoder technology.

## 6.1 Application window behavior on the Windows OS

In the Windows OS, a window consists of background layers and overlay layers. The overlay layers, as the name suggests, lay atop the background layers. There are two types of windows in the Windows OS namely, a normal window and an overlay window. When the OS triggers a paint event, normal windows mostly render/paint on the background layer while on the other hand, overlay windows paint mostly on the top layers.

The color of a pixel is a determined by the values of its Red, Blue, Green components and an alpha-component value. The alpha-component value defines the opacity of a pixel. The alpha-component of a pixel can take values between 0 and 255, with a value of zero indicating that the pixel is completely opaque and a value of 255 indicating that it is transparent and colorless. Thus, manipulating the alpha-component values of all pixels constituting a window to higher values (nearer 255) gives a see through glass effect enabling the creation of transparent and semi-transparent forms. Since the pixels are transparent or semi-transparent on such forms, one can see the underlying windows or user’s Windows desktop through them. Pixels of general application windows on Windows OS usually have a value of zero for the alpha-component, making them opaque. Application windows like Word®, Adobe Acrobat®, Excel®, Journal® and so forth are examples of opaque windows/normal windows through which one cannot see the background windows or desktop. On the other hand, an overlay window has a high alpha-component value for all of its pixels and hence one can see application windows underneath it.

Programmatically, the value (in percentage) of the opacity property determines value of the alpha component of pixels in window/form. As the name of the property suggests, an opaque form has the opacity value of 100%. A transparent window/form has opacity of 0%. Semi-transparent or translucent forms like the email notification popup form of the Microsoft Outlook have an opacity value that is usually in the range of 10 to 70.

WindowsMedia Player application window is an example of an overlay window that paints on the top layer rather than in the background layer. This is because WindowsMedia

Player uses the DirectShow video renders that usually render video on an invisible overlay surface. WindowsMedia Encoder technology used to implement the screen capture functionality does not capture such special overlay windows. Hence, in order to record a movie file and the ink annotations made on it using WindowsMedia Encoder, it was necessary to design and develop the solid eVellum. The solid eVellum is a normal transparent borderless window through which one can see such overlays. The details of the behavior of the solid eVellum can be found in [2].

Since we only use the DirectX technology for screen capture in WriteOn1.0, we elaborate on the Overlay Vellum in Section 6.2.

## 6.2 Overlay vellum

WriteOn1.0 implements the overlay vellum since it implements its screen capture and broadcast using DirectX technology. The overlay vellum integrates well with the Video for Windows(VfW) framework used to record screen activities to a AVI files (Chapter 7) in WriteOn1.0. The overlay vellum consists of a transparent borderless window and another almost transparent borderless window, both of the same size lying one over the other. One of the windows is the child overlay window and the other is the parent overlay window. The parent overlay window lies behind the child overlay as seen in Figure 6.1.

The parent window is a completely transparent and borderless form through which a user can see the underlying background. All the pixels in this form have the alpha-component value set to 255 making them transparent. Programmatically, we can manipulate the Transparency Key property of Windows forms to achieve the transparent effect. The Transparency Key of a Windows form indicates which colors appear as transparent. In order to achieve a see-through effect, we set the Transparency Key's attribute value to *Background* color. This means that if a pixel on the window is that of the background pixel color, the window renders it as transparent. If all the pixels making up a window are set this way, the parent window becomes transparent. The parent window cannot display ink annotations because any stroke on the window would be invisible because of the value of the Transparency Key. In addition to that, because of the way Windows forms and the Windows OS behave, all of the mouse or stylus events on the transparent region of any form logically pass through them to the underlying applications, which

receives the events instead of the transparent parent window. In order to be able to write and render ink, we use a second window atop the parent window called the child overlay.

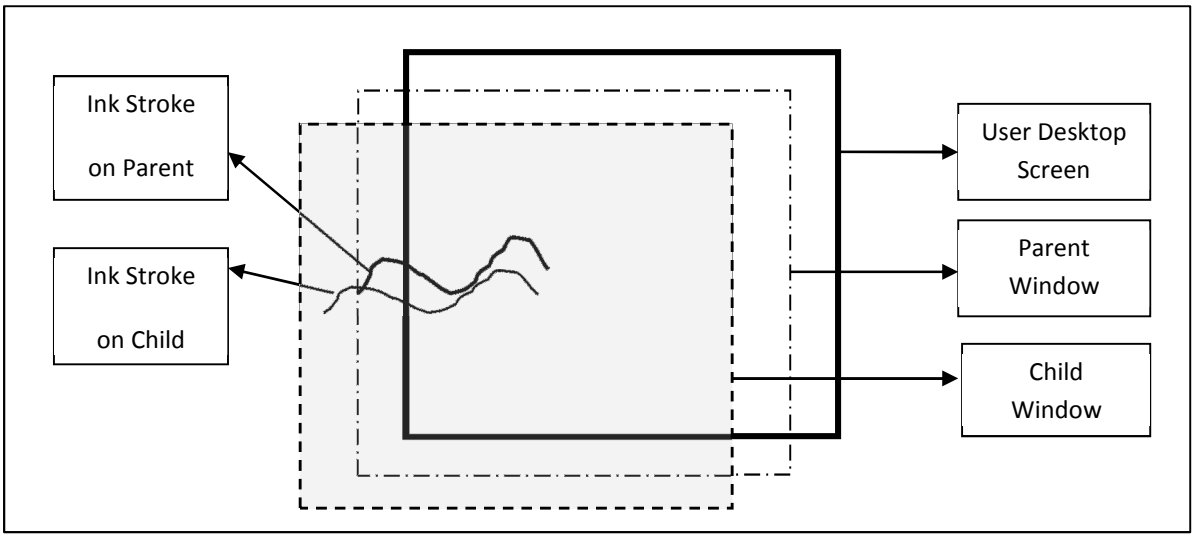



**Figure 6.1: Illustrating child window overlapping the parent window over Desktop**

The child overlay window is a translucent window with its opacity value set to 0.05%. As shown in figure 6.1, it resides on top of the parent window. Due to the way Windows GDI and Windows OS behave, the opacity of the ink strokes depend on the opacity of the form thus making the ink on the child overlay only 0.05% opaque or nearly transparent. In order to achieve opaque ink strokes, we render all the ink strokes already received by the child window on the underlying parent window. The parent window renders dark, solid and opaque ink strokes on it. Since both the forms have the same size and location, the ink strokes appear at the same location on both the forms producing a non-distorted opaque ink stroke on the eVellum as illustrated in Figure 6.1.

Microsoft.StylusInput class provided by Tablet PC SDK handles events received from the digitizer of the Tablet PC screen and achieves the ability to ink on the eVellum. Section 6.3 explains the technical details on how annotating and erasing ink are achieved on the eVellum.

## 6.3 Working of the stylus

Section 6.3 explains in detail, the mechanism of annotating, retaining and changing ink attributes on the eVellum in WriteOn1.0. The RealTimeStylus API detailed in sub-sections 6.3.1 and 6.3.2, realizes electronic pen strokes on the Tablet PC screen to ink strokes on the eVellum. The user can toggle or deactivate the eVellum at any point of time while using WriteOn1.0. All the ink annotations made before the deactivation event are retained and rendered upon reactivation of the eVellum. Sub-section 6.3.3 explains how WriteOn1.0 retains all the ink upon toggling the eVellum. The last sub-section, 6.3.4 details how the eVellum of WriteOn1.0 maintains a uniform pen width when the user applies varied pressure while writing

### 6.3.1 The RealTimeStylus API

The RealTimeStylus class of Microsoft.Ink DLL provides functions to access the Tablet PC pen events and pen data stream in real time. The RealTimeStylus API functions act as an interface to digitizer screen and collects stylus packet data. The RealTimeStylus object process the pen data and passes it to the ink collector object of the eVellum.

In order to interact with the pen data, one has to add plug-ins to the RealTimeStylus object. There are two types of plug-ins namely; synchronous plug-ins and asynchronous plug-ins. Synchronous plug-ins are added for computationally un-intensive tasks that do require access to real-time pen data, such as filtering of packets. Asynchronous plug-ins are used when we do not require real-time access of the packet stream, such as creating ink strokes and passing it to an ink collector object.

In order to process the stylus packets, the RealTimeStylus object of the Microsoft.StylusInput API internally maintains an input queue and an output queue. As soon as the digitizer of the Tablet PC detects the pen stylus, stroke packets are generated and stored in data structures, which are objects of the Microsoft.StylusInput.PluginData class. The stroke data objects are added to the input queue as they are generated and contain all the pertaining stroke information. The stroke information comprises of the details of the stroke like the location of the real-time cursor in ink space coordinates, whether the stylus is inverted, pressure on the pen tip,

coordinates of points making up the stroke, color and width. The RealTimeStylus API can delete strokes, modify the stroke data or add strokes before sending the packets to the output queue.

The IStylusSyncPlugin interface of the SyncPluginCollection class actually implements the code to process the stroke packets. Each of the plug-in data objects are sent to the RealTimeStylus object's synchronous plug-in object collection. The synchronous plug-in object can modify the input and output queue by adding data to either of them. The plug-in objects move to the RealTimeStylus object's output queue after sending the packet objects to the synchronous plug-in collection while the RealTimeStylus listens for the next plug-in data object to handle.

The RealTimeStylus object also can send one plug-in data object from its output queue to the objects in its asynchronous plug-in collection. Like the synchronous plug-in, each asynchronous plug-in can add data packets to both the input or output queue. However, since the asynchronous plug-ins run on the UI thread, any packet data that is added would be in relation to the to the current pen data being processed as opposed to the data that the asynchronous plug-in is processing.

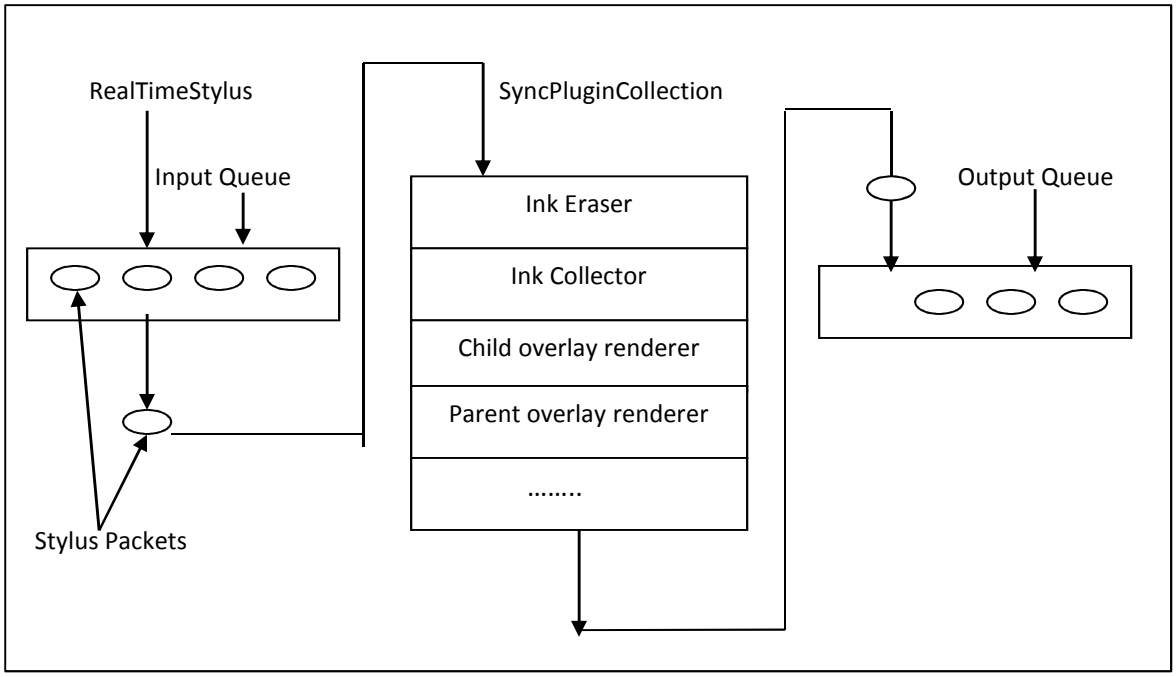



**Figure 6.2: Work flow illustration of the RealTimeStylus**

The InkCollector and DynamicRenderer examples of classes that implement the IStylusSyncPlugin interface. As seen in Figure 6.2, these plug-in objects receive the stroke packets from the input queue and they pass them after processing to the output queue. Unlike the asynchronous plug-ins, the synchronous plug-ins can directly access the RealTimeStylus queue and need not necessarily run on the UI thread. The synchronous plug-in objects contain the ink collector objects that gather and render ink. The next sub-section explains the rendering and erasing of ink strokes on the eVellum.

### 6.3.2 Ink annotations using RealTimeStylus API

The dynamic-renderer object of the eVellum renders ink strokes as the user draws them. The parent and child overlay windows of the eVellum have a dynamic-renderer object associated with them. The dynamic-renderer is a synchronous plug-in that implements the IStylusSyncPlugin interface which displays the pen data, like strokes, while they are being drawn in real-time.

The InkCollector object collects electronic ink form the Tablet PC hardware and sends it to the application. It essentially acts as a container of all the ink strokes the user makes on the eVellum before it passes them to the dynamic-renderers for rendering. The Ink object caches the electronic ink as soon a stroke is completed. In order to ensure we render the same set of strokes on both the parent and child overlay windows the Ink object is shared between them. Each of the overlay window's dynamic-renderer objects draws the ink strokes present in the Ink object based on their packet information like location, pressure, color, width, etc., enabling the user to annotate on the eVellum.

The user can erase the ink strokes with the back of the stylus, just like a normal eraser behind a pencil. The InkEraser object is a user defined synchronous plug-in implemented in WriteOn. When the digitizer detects the back of the pen stylus, it disables the InkCollector from collecting ink and enables the InkEraser object. In the eraser mode, the stroke(s) that intersect the point at which the eraser tapped the screen are erased from the eVellum. In the eraser mode, any incoming stroke packets are deleted from the internal RealTimeStylus queue so that the child and parent renderers do not paint them.

The first implementation of the eVellum lacked the feature to retain ink on it during a deactivation followed by a reactivation. This meant that any ink on the eVellum was deleted

when the user toggled the vellum to make it invisible. Thus, the user was unable to view previous annotations upon toggling the eVellum again to make it visible again. The scalable eVellum of WriteOn1.0 preserves the strokes when the user toggles the eVellum and makes it visible again. The next sub-section explains the importance of this feature in a classroom environment and how WriteOn1.0 incorporates this feature into the scalable eVellum.

### 6.3.3 Toggling and retention of ink on scalable eVellum

In this sub-section, we shall provide reasoning behind the implementation of the ink retention feature in the scalable eVellum. As mentioned earlier, the first eVellum implementation lacked ink retention because of which it deleted all ink in its Ink object when it was toggled to make it invisible. When the user toggles the eVellum to make it visible again, a new eVellum without any ink appears to him/her. This scheme works well when the instructor uses a PowerPoint slides and wants a new eVellum as he/she moves to the next slide. However, this mechanism does not prove useful in most classrooms especially those involving subjects like circuit analysis.

Instructors who teach subjects like Digital Signal Processing, Electronic circuits use simulation software to explain subject matter from a practical point of view. In such classrooms, the instructor often addresses a "what-if" question. An example of a “what-if” would be “what happens to the current in the circuit if the voltage is increased?” Such “what-if” questions may need the instructor to make certain parameter changes in the running simulation and re-run it to view the results. Because the eVellum was enabled over the entire desktop area in the early version of the WriteOn tool, the instructor had to first toggle the eVellum, make the necessary changes in parameters and enable it back again to resume explaining the topic. As a follow-up to the “what-if” question, a comparative analysis of the results of the first and second simulation may be of interest to the subject matter in hand. In order to perform a comparative analysis between the results of the first simulation and the second, the instructor would most likely need the notes he/she had noted down while running the first simulation.

Such comparative analysis is impossible when the eVellum does not preserve the ink when toggled to deactivate it, as is the case in its first implementation. In addition to performing comparative analysis, preserving all the ink annotations made during the course of a lecture helps a lecturer to summarize all the main points in the end of the class. Thus, it becomes imperative

that we preserve and store all the ink strokes until the user initiates a clear all ink action on the eVellum.

WriteOn1.0 preserves all the ink strokes by storing them in an instance of the Ink object associated with the scalable eVellum every time the user toggles the eVellum to make it invisible. When the eVellum is toggled back to its visible state, all the strokes stored in the Ink object instance are rendered. The scalable eVellum of WriteOn1.0 supports four pen width settings and an array of colors. The final sub-section lists the supported pen widths in WriteOn1.0 and explains how the eVellum renders the strokes at a set thickness irrespective of the pressure applied on the pen stylus.

### 6.3.4 Setting a fixed width of pen

The width of the pen stroke is measured in himetric units. One himetric unit is one thousandth of a centimeter and is independent of the screen resolution. The eVellum in WriteOn1.0 supports four pen widths. The four widths are default, narrow, medium and thick. Their measurements are 100, 60, 130 and 150 himetric units respectively.

One of the ways to draw strokes of varying thickness is to apply pressure on the pen tip while drawing the stroke. The dynamic-renderer renders more ink on the y-axis of the computer screen, drawing a thicker stroke when pressure is applied on the pen stylus, provided the IgnorePressure value is set to false. As the name suggests, the IgnorePressure attribute when set to true, does not take into account the pressure applied on the pen tip, rendering a stroke at a prefixed width irrespective of the pressure applied by the user. By default, when maximum pressure is applied, the stroke width is 150% of the set width value and when minimum pressure is applied, the stroke is drawn at 50% of the width value set.

Thus, in order to achieve strokes which have a fixed width, the IgnorePressure field is set to false for the parent and child renderers of the scalable eVellum. Disabling IgnorePressure ensures that the renderer shall render ink strokes at the specified width irrespective of the pressure applied on the pen tip.

The scalable eVellum is called so because, unlike its first version, it can be resized and moved like any form/window on the user's Windows desktop. Sections 6.4 and 6.5 explains the technical details and the significance of creating a resizable and movable eVellum in WriteOn1.0

called the scalable eVellum, to differentiate it from the eVellum of the first version of the application.

## 6.4 Resizing a transparent borderless form (eVellum)

The eVellum implementation in WriteOn1.0 is called the scalable eVellum. One of its highlighted features is its flexibility in terms of moving and resizing it. As opposed to the static eVellum of the first version of WriteOn, the scalable eVellum takes the dimensions and location defined by the user. Since its dimensions can be resized and its location can be changed during the course of a user session, it can be used to highlight an area of interest. Like any other window on the user's desktop, the user can resize the scalable eVellum using either the mouse or the pen stylus. Sub-section 6.4.1 details the technical challenges of implementing a resizable eVellum and explains how the eVellum was implemented in WriteOn1.0. Sub-section 6.4.2 looks at how a screen recording and broadcasting session are affected by resizing the eVellum.

### 6.4.1 Creating the resizable eVellum

The first step in creating a truly scalable eVellum is to make it resizable. The user must also have the option of resizing the eVellum using the pen stylus when in tablet mode. In technical terms, the overlay eVellum is composed of two borderless forms: one transparent and the other one almost transparent, of the same dimensions and coordinates one on top of the other. Thus, in order to create a resizable eVellum, we need to create resizable transparent borderless forms. However, due to the design of Windows forms, forms that lack a border cannot be resized and relocated. This sub-section explains the details on how WriteOn1.0 realizes the resizable eVellum despite the restrictions on resizing and moving a transparent borderless form.

One of the simplest ways to creating a resizable eVellum is to create a border around it by changing the FormBorderStyle property value of the parent window from *none* to *sizable*. The user can now begin to resize the eVellum like any other form/window. The child overlay window can be re-dimensioned to the same dimensions as that of the parent overlay window by detecting the size change event in the parent form and applying the same size dimensions to the child window. Even though this implementation produces the desired resizable eVellum, it is not a very practical solution from the perspective of user experience. Since the eVellum is transparent, one of the drawbacks of this scheme is that the borders are not markedly visible on the user's

Windows desktop. This makes it difficult for the user to identify and locate the eVellum border for resizing, rendering this method of creating a resizable eVellum on its own almost useless. As we shall see in the paragraphs to follow, this method is first step in creating a resizable borderless form. When this mechanism is augmented with other schemes, we can create the resizable scalable eVellum.

The other drawback is the lack of a title bar in the eVellum window. The absence of the title bar in a Windows form does not permit the user to drag the form to a different location. The other major drawback is that since the borders do not count as a part of the Windows form's client area, we cannot detect any mouse events like mouse-down, mouse-up and double-click. The Windows OS detects a resize event on an application window by capturing the mouse events when the mouse cursor is at or near the window borders. The order of mouse events that occur during a resize event are pressing the left-click button of the mouse on the border followed by a drag operation of the cursor to a new location of that border and then finally release of the mouse cursor from the border. This causes the window to change the location of the border which is dragged, that in turn expands or contracts the window. However, due to the inability to of detect mouse activities on the form borders, we cannot programmatically determine when the user has started or stopped an eVellum resize by creating a form with resizable borders. This information is necessary because the area of the screen captured for broadcast and recording purposes and is the same area as that of the scalable eVellum. Thus, we need to update the area of screen capture and broadcast as soon as we detect a change in the dimensions and coordinates of the eVellum. Sub-section 6.4.2 explains the effect of resizing the scalable eVellum on the screen capture and broadcast session.

In the Windows OS, an application window can capture all the mouse events that occur in its client area only. However, since borders do not count as a part of the client area in the Windows OS, we cannot detect the mouse actions which define the start and stop of a resize event on a window by default. Moreover, in the case of the scalable eVellum, since the parent window (resizable) is transparent, all the mouse events that occur on its client area like click, drag and release logically pass to the underlying screen/window. Thus, a mechanism to capture the mouse events on the overlay window had to be devised. For this purpose, the red border is drawn in the client area of the transparent parent overlay window making the border pixels

opaque. The new border can now receive and detect mouse events since it is in the windows client area.

The way this scheme works is, whenever the user places the mouse cursor on the border of the scalable eVellum, we detect a mouse down event since the user would click in the windows client area. From the position of the cursor, we can determine if the cursor is at the border by computing its distance from the edges of form. If the cursor is at a distance less than five pixels away from edge, we consider that the mouse is at the border and that a potential resize event may occur. We shall call this the resize mode. Since the parent overlay has a sizable border, the Windows OS handles the implementation of the resize. The end of a resize operation can be detected by capturing the change in the mouse location and the mouse-up event at the red border. These events are sent as predefined messaged to the application window. The Appendix II gives the code snippet used to achieve this feature.

The WndProc function, which is an application defined function, handles the processing of messages sent to a window. It takes as an argument, a reference to the message object. The message object contains the details of the message sent to the window. The details can be the mouse position if the message is a mouse event or windows messages like a close window or maximize a window. Each of these messages has a special value which identifies them. For example, a mouse click on the client area has the value of 0x01 and it known as the `HTCLIENT` message. The WndProc method can perform special functions like resource cleanup when a `WM_DESTROY` message is receive. The `WM_DESTROY` message is sent when the user clicks on the close button of a window or uses the Alt+F4 key combination. The WndProc function can intercept the messages coming to the window and perform specific tasks. The code implemented in the WndProc function (Appendix II) of the parent overlay window is the brain of the eVellum that helps process the mouse events to transform the static eVellum from single sized transparent form(s) to a resizable one.

Essentially, the WndProc function processes the messages sent to or from the application in the following order: it first detects the position of the mouse cursor in order to determine a potential eVellum resize event. From the dimensions of the eVellum and the mouse cursor position, we compute to see if the mouse cursor is at the border of the eVellum. Now, if the user clicks on the border of the parent overlay of the eVellum, a `WM_NCMOUSEDOWN` message is sent to the eVellum and we set the resize flag to true. Upon receiving the `WM_NCMOUSEDOWN`

message, since the parent window has a sizeable border, the Windows OS resizes the eVellum window as the user drags one of the borders on which the mouse cursor is present. The WndProc function sets the resize mode to false as soon as it detects a mouse-up event on the same border. The next sub-section illustrates how a resize of the eVellum affects the screen capture session.

### 6.4.2 Resizing the eVellum during screen capture

The scalable eVellum can be resized like any normal window using a mouse or the stylus. WriteOn1.0 maintains consistency between the area over which the eVellum is enabled and the area which is captured. In other words, once a user selects an area of interest to capture screen activities either to a file or for broadcast, the scalable eVellum is enabled only in that selected area if the user decides to annotate. This also implies that when the user chooses an area over which to annotate, that area becomes the area of interest by default during a broadcast or screen recording session. In other words, we are essentially resizing our area of interest when we resize the eVellum. Thus, we need to provide mechanisms by which we reflect the change in area of interest in the screen capture and broadcast process.

One of the challenges is that, the functions which stream the compressed video data to an AVI file do not support frames of different sizes be written to a single AVI file. This means that we cannot re-dimension the eVellum, change the capture dimensions and continue recording to the same AVI file. Thus, we need to stop the current recording session, close the AVI file, cleanup resources and create a new AVI file to start capturing activities in the new area, each time the user resizes the scalable eVellum. It is very common for a user to resize the eVellum multiple times during a lecture session. In order to string together all the AVI files created in a lecture session, we create a playlist of AVI movie files. The user determines the name and path of the playlist. When the user resizes the eVellum, the current screen recording session stops, the AVI file is saved at the user specified location and the file is finally appended to the user specified playlist. The capture parameters are modified to reflect the values of the new eVellum dimensions followed by the screen recording being restarted with the new parameters.

The name of the playlist, identified by the CourseID attribute value received as a user input, acts as the prefix to the AVI files created each time the user resizes the vellum during a recording session. If the students and instructor want to save all the lecture notes of a semester as a single playlist, during the first lecture, the user would create the new playlist, say in his/her

lecture folder and records the first lecture to the playlist. The instructor or student can choose to append the subsequent lecture sessions into the course playlist, instead of creating a new playlist for each lecture session. Since all the AVI files in this case shall have the same prefix, it is important that we follow a naming scheme that shall ensure unique names to the AVI files created, even when the prefix is same, like in the case of the same course ID. The AVI files follow a naming scheme that has course ID as prefix followed by the count of the AVI file and then the date of creation and finally time stamp of creation. This shall ensure that there are not two AVI files created with the same name.

When the user stops a screen capture session, the current AVI file to which he/she is recording is closed, saved and appended to the playlist, after which the playlist is closed and saved. During a lecture broadcast, after a scalable eVellum resize event, the broadcast is stopped. The new capture parameters are applied and the broadcast session resumes. The next section takes a look at how we achieve the ability to move the scalable eVellum in the absence of a title bar.

## 6.5 Moving the eVellum

Another aspect to making the static eVellum dynamic is the ability to drag and drop it from one location on the screen to another. Such an ability to drag and move the scalable eVellum on the user's Windows desktop shall give the user much needed flexibility while using WriteOn1.0. This means that the user can just move the scalable eVellum and not toggle or resize it to access any underlying window or desktop. The sub-sections 6.5.1 and 6.5.2 explain the challenges and implementation details of the movable eVellum.

### 6.5.1 Creating the movable eVellum

The final step in creating the scalable eVellum is to make it movable in the sense that a user can drag and drop the eVellum from one point to another on the screen. However, since the eVellum does not contain a title bar, one cannot drag and move it on the user's Windows desktop like any other window. This obstacle can be overcome by tricking the Windows OS to believing that the user has clicked on the title bar of a window when the user clicks inside the border of the new eVellum simply by sending the `HTCAPTION` message to the eVellum window. When a window receives this message in its WndProc function, it fires the move window event that

enables us to move the eVellum. The first step is to detect if the mouse is at a point inside the edges of the red border based on the location of the mouse cursor and the dimensions of the eVellum. The next key step is to capture the `WM_NCHITTEST` message. This message has a value of 0x64 and is sent to a window when the mouse is moved over it or a mouse click is made on it. We then intercept the `WM_NCHITTEST` message while the mouse is at the border of the eVellum and return the `HTCAPTION` message to the parent overlay window. The `HTCAPTION` indicates that the mouse action took place on the title bar. The `HTCAPTION` triggers a move event on the window, thereby relocating the eVellum. It is very important to verify that the mouse action is on the scalable eVellum's custom red borders only. Otherwise, any mouse down event including those for inking purposes will be result in a scalable eVellum move action.

The next sub-section explains how moving the eVellum affects a screen capture or broadcast session.

### 6.5.2 Moving the eVellum during a screen capture session

The scalable eVellum's location can be changed using the mouse or the pen. This change of location alters the coordinates of the area of interest. Since the area of interest and the screen capture area are linked in WriteOn1.0, we need to change the capture parameters. This is simple since the size of the area of capture does not change. Thus, we only need to update the top and left coordinates of the capture area. There is no need to stop and resume the screen capture process to reflect the new location of the eVellum.

Now that we can move and resize the eVellum, we shall explore the various modes of operation of the scalable eVellum through which a user can write on and toggle it. There are three primary modes of operation of the scalable eVellum while using WriteOn1.0. These modes are the vellum mode, pierce mode and the transparent mode. Section 6.6 explains each of these modes in detail.

## 6.6 Modes of eVellum in WriteOn 1.0

The modes of operation of the scalable eVellum represent the various visible states at which the eVellum can exist. The eVellum can be operated in three modes namely, the “Vellum mode”, “Pierce mode” and the “Transparent mode”. In the Vellum mode, the scalable eVellum is

enabled and visible. The eVellum is inactive or invisible in the pierce mode. While the vellum mode and pierce mode were present in the first version of the eVellum, WriteOn1.0 introduces a new mode of operation of the eVellum called the transparent mode. In this mode, the user can access the application underneath the eVellum without having to toggle the eVellum. The sub-sections 6.6.1, 6.6.2 and 6.6.3 further elaborate on these modes of operation.

### 6.6.1 Vellum mode

In the vellum mode, the eVellum is enabled and active over a user-selected area. The eVellum is the top-most window on the user desktop which resides over all the current application windows. The user can write on the eVellum using the pen and erase ink strokes using the back of the stylus. The user can flip the pen back to resume writing. The user can change the pen color and width by choosing the appropriate menu option from the WriteOn1.0 toolbar. The user can wipe the eVellum clean of all strokes by clicking the "erase all strokes" menu item from the toolbar. The user can move and resize the eVellum in this mode of operation.

### 6.6.2 Pierce mode

The pierce mode is an important and essential mode of operation of the eVellum. As the name suggests, the user pierces the eVellum, making it invisible to access the underlying application or desktop. The pierce mode is crucial to an active learning session, where an instructor can access any application or window on the desktop which may contain lecture material. The instructor can then re -enable the eVellum over the new window to illustrate the new topic.

The user can pierce the eVellum by clicking the same menu button on the WriteOn1.0 toolbar from which the eVellum was enabled. The menu button is highlighted to lime green to indicate that the eVellum is enabled. The eVellum is pierced by clicking on the highlighted "enable vellum" menu button. Upon doing so, the highlight over the menu item is lost, indicating that the eVellum is disabled. The user can also disable the eVellum by using a hot key combination (Alt + F2) to pierce the vellum. The same key combination toggles the eVellum back to vellum mode. This hot key combination is preset when WriteOn1.0 is installed. The toggling effect of the eVellum using the hot key is achieved by the Windows hooks mechanism.

The eVellum retains all the ink strokes on it when it is pierced and renders them all when the eVellum is re-enabled.

When in tablet mode, the user navigates the desktop using only the stylus. Thus, it is desirable to execute the context switching with the same ease as in the PC mode. The option to toggle the eVellum with a single click of a button in WriteOn1.0 is an important usability feature, allowing the instructor the same flexibility of use of the application in the tablet mode as that on the PC mode.

Most Tablet PC's have special buttons called Tablet buttons which are located on the display panel. These buttons can be mapped to any action the user may want to perform like starting an application, reducing the speaker volume, keystrokes or combination of keystrokes. The default settings of these keys vary from Tablet PC to Tablet PC. The user can map one of the Tablet buttons to Alt+F2 before starting WriteOn1.0 for the first time. The user can then toggle the eVellum by pressing the special button when in the Tablet mode.

The user can always move the scalable eVellum or resize it to access an area on the desktop beneath it.

### 6.6.3 Transparent mode

The third mode of operation of the eVellum is the transparent mode. This mode of operation is one of the most compelling features of WriteOn1.0. In this mode of operation, the user can access the underlying application window/desktop without making the eVellum invisible or changing its location. This means that the user can access and open a folder by clicking "through" the eVellum by the right-click of the mouse, thereby eliminating the need to disturb the current state of the eVellum and its ink. This feature comes very handy when illustrating a simulation and performing comparative analysis. The instructor can access the simulation control buttons like the start and stop buttons by just right clicking on the eVellum. This exposes the underlying application/desktop as if there was no vellum over it for the user to perform his/her tasks while the user is able to see the ink annotations intact.

By allowing the instructor to access any application controls and the desktop with a click of the mouse, we give the user the ease and speed of context switching. In addition to that, by preserving the ink annotations, we allow him/her to remain in the current context of lecture presentation.

The transparent mode of operation is implemented by exploiting the feature of transparent parent overlay window which logically passes any mouse events performed over its transparent regions to the underlying desktop. When the user right-clicks on the eVellum, technically, the child overlay window is made invisible while the transparent parent window is still "visible" with all the ink annotations. The WndProc method listens for the right click mouse event and performs the action of making the child overlay window invisible. The user can reactivate the pen cursor for writing by placing the mouse cursor over the red border of the scalable eVellum. This mouse enter event sends the `WM_NCHITTEST` message which is processed in the WndProc function to toggle the child vellum back to its visible state for the user to start writing again. Appendix I contains the WndProc function of the overlay vellum that implements the transparent mode of operation of the scalable eVellum.

The ScreenCaptureto AVI module captures all the activities in the area of the scalable eVellum to an AVI file. Chapter 7 explains the screen capture as AVI file module of WriteOn1.0. This module was developed using the VfW API.

# Chapter 7

# Screen Capture to AVI

WriteOn tool provides functionality to record a composite of the screen activities and ink annotations on the eVellum to an AVI file. The Monitor Device module implemented this feature in the first implementation of the application. With the implementation of WriteOn1.0, the screen capture as AVI file functionality has been moved from the Monitor Device module to a new module called the ScreenCapturetoAVI module.

In the first implementation of WriteOn, the two operations of recording the lecture to an AVI file and broadcasting it were mutually exclusive in the sense that the instructor either could broadcast a lecture or could save it to a file. There was no provision to accomplish both these tasks simultaneously. The lack of this feature meant that the instructor cannot make a lecture presented today available to the students immediately because he/she was unable to record it as a movie file while broadcasting it. In order to facilitate the instructor to broadcast and record a lecture presentation in parallel in WriteOn1.0, a module that shall exclusively perform the function of saving screen activities to a movie file was developed. The new module in WriteOn1.0 is the ScreenCapturetoAVI module. Deployed as a COM DLL, this module is implemented in C++. The ScreenCapturetoAVI object exposes the record to AVI file function. After instantiating the class in WriteOn1.0, the application calls the record function which starts writing the screen capture frames to an AVI file. The ScreenCapturetoAVI capability is implemented using the Microsoft® Video for Windows (VfW) framework.

The VfW provides a wide array of functions to create an AVI and write to it. Microsoft® Video for Windows (VfW) provides functions that enable an application to process video data. The AVIFile functions and macros handle the information in an AVI file as one or more data streams instead of tagged blocks of data called chunks. Data streams refer to the components of a time-based file. An AVI file can contain several different types of data streams interleaved together such as a video sequence, an English soundtrack and Spanish subtitles. Using the

AVIFile functions, one can access and modify each of the streams comprising an AVI without affecting the other streams.

In order to open an AVI file, write frames to it and then close and save it, we need to follow the steps listed below:

i. Initialize the AVIFile library.
ii. Open the AVI file for writing.
iii. Capture a sample frame and check its compatibility with the compressor or codec. This is because some codecs have special requirements like even dimensions.
iv. Fill the Video stream header details like the stream type, buffer size, frame rate, etc.
v. Create an AVI stream for the file.
vi. Set the compression options like the codec information, key frame rate, etc.
vii. Create a compressed AVI stream from the previously created data stream and compression options.
viii. Start writing the compressed stream to the AVI file.
ix. Close the AVI file upon termination of the record session.
x. Free all the pointers and call AVIFileExit function

The VfW framework provides various the necessary functions to achieve the steps listed above. Because of the speed and compression ratio offered by the MSU Lossless Screen Capture codec, WriteOn1.0 uses it to create a compressed AVI file.

The rest of this chapter describes the functions used to implement the screen capture to AVI file functionality.

## 7.1 Initializing the AVIFile library

AVIFile functions and macros are contained in a dynamic link library. The AVIFile library has to be first initialized to access any of the AVIFile functions. To initialize the library, use the *AVIFileInit* function. After the initialization of the library, one can use any of the AVIFile functions or macros. Before we terminate a record session, we have to release the library. To release the library, use the *AVIFileExit* function. AVIFile maintains a reference count

of the applications that are using the library, but not those that have released it. WriteOn1.0 balances each use of *AVIFileInit* with a call to *AVIFileExit* to completely release the library after terminating the record session. Section 7.2 explains the function to open an AVI file for writing.

## 7.2 Opening an AVI file for writing

The *AVIFileOpen* function opens an AVI file and returns an address of the pointer to a buffer that receives the new *IAVIFile* interface pointer, if the function succeeds. The function has the following arguments:

- The pointer to a buffer that receives the IAVIFile interface pointer
- Name of file to open
- The mode in which the file has to be opened like CREATE mode and WRITE mode in addition to the default READ mode
- Pointer to a class identifier of the standard or custom handler you want to use. This value is set to NULL meaning that the system would choose the appropriate handler from the registry.

The function returns zero if it succeeds. Once the AVI file open function succeeds, the next step is to validate the MSU screen capture codec. The compressor validation process is explained in the following section.

## 7.3 Validating the compressor

In order to validate a compressor, we first need to capture a sample frame. In WriteOn1.0, a sample screen shot of size 320 X 200 is captured using the WindowsGDI functions as seen in Appendix II. After capturing the sample frame, the validity and compatibility of the MSU Screen codec v1.2 is tested. There is a need to conduct a validity test because some compressors, for example the MSU screen codec v1.1, call for special conditions on the uncompressed frames before it can compress them. An example of one such special condition for the MSU screen codec v1.1 is that, it can only compress frames that have even dimensions. Another example of such pre-conditions on the uncompressed frame is the need for the dimensions of the frame to be a multiple of eight.

In order to test the validity of the codec, we try to open the desired compressor, uniquely identified by its FourCC code, from a list of video codecs installed on the system on which WriteOn1.0 is running. A FOURCC (literally, four-character code) is a sequence of four bytes used to uniquely identify data formats. One of the most well-known uses of FourCCs is to identify the video codec used in AVI files. The FOURCC code of the MSU Screen Capture codec is *SCLS*.

We procure a list of compressor information structures that hold data related to the video compressors by using the *ICInfo* function. By passing the *ICTYPE_VIDEO* as the first argument, we get an array of video compressor information structures. From this array of information structures, we can get a pointer to the information of the MSU screen codec v1.2 by its FourCC code. From the list of the compressors, we choose the MSU Screen Capture codec v1.2 by first opening the Compressor using *ICOpen* function.

The *ICOpen* function opens a compressor or decompressor. Since we already have information about the video compressors, we open each compressor information structure in the list in query mode to get information about it. Using the information fetched about the compressor, we can verify if the codec has the same FourCC code as that of MSU screen codec v1.2 from the fccHandler attribute of the ICInfo structure that stores the information of the codecs.

Once we are able to find the ICInfo structure associated with the MSU screen codec v1.2, using the *ICCompressQuery* function, we query if the compressor is compatible with sample frame captured. Since the MSU Screen Capture codec v1.2 has no special requirements, this test always passes. The next step is to fill the information related to the data stream that is going to be written to the AVI file.

## 7.4 Filling the AVISTREAMINFO structure

The AVISTREAMINFO contains information for a single stream. Some of the attributes are the type of stream, the FOURCC code of the stream, quality, buffer size, dimensions of the captured frame among other attributes. The following code snippet gives the values for some of the attributes that are important:

fccType = streamtypeVIDEO;

fccHandler = *mmioFOURCC* (‘S’,’C’,’L’,’S’)

```
dwScale             = fps;
dwRate              = fps;
dwSuggestedBufferSize  = userframe->biSizeImage;
```

Here, the *mmioFOURCC* is an inbuilt macro that converts four characters, like the ones that identify a codec format, uniquely into a four-character integer code. The *dwScale* gives the time scale applicable for the stream. Dividing *dwRate* by this value gives the playback rate in number of samples per second. For video streams, this rate should be the frame rate. The *dwSuggestedBufferSize* attribute is the recommended buffer size, in bytes, for the stream, which can be set as the size of the image of the captured frame that has dimensions of the capture area. If we are not sure of the frame size, the value of the *dwSuggestedBufferSize* attribute is set to zero. The next step is to create an AVI stream associated with the file based on the AVISTREAMINFO structure.

## 7.5 Creating the AVI stream

In order compress the data before writing to the AVI file, we need to create a compressed video stream associated with an AVI file. In VfW, a compressed stream can be only created from a normal AVI stream. The *AVIFileCreateStream* function takes as arguments, handle to an open AVI file and pointer to the AVISTREAMINFO structure and creates a new stream in the file and returns the pointer to a new stream interface. The AVISTREAMINFO structure stores the data for the type of the stream and frame rate. The AVI file has to be created with write permission to be able to create the AVI stream successfully. This function returns zero if it is successful.

## 7.6 Creating Compressed stream

In order to utilize optimum memory for storing the AVI file, we need to compress the video stream. The *AVICOMPRESSOPTIONS* structure contains information on how a stream is compressed and saved. The attributes like fccHandler and buffersize have the same value as that in the AVISTREAMINFO structure. The data in this structure is used by *AVIMakeCompressedStream* function which creates a compressed audio/video stream from an uncompressed stream and a compression filter. The *AVIMakeCompressedStream* returns a

pointer to the buffer that receives the compressed stream pointer. The compression filter is the codec and it is the MSU screen codec in WriteOn1.0. This function returns zero if it is successful.

## 7.7 Writing to AVI file

The *AVIStreamWrite* function writes the captured frames to a stream. The function takes as input the handle to stream, the time at which the frame is written and the dimensions of the rectangle representing the frame of data among other parameters. Since we are writing a compressed data stream to the AVI file, the handle passed to this function would be the handle to the compressed video stream. The ScreenCapturetoAVI module treats each frame written independent of the previous frames. The function appends data frames to the end of the stream. The function overwrites the data in the stream pointed to by the handle passed to the function. AVIStreamWrite function returns zero if it succeeds.

## 7.8 Stopping a record session

All the resources like stream pointers and file pointers have to be freed once the recording session is terminated. The *AVIStreamRelease* function decreases the reference count on a stream and closes the stream if it reaches zero. The *AVIStreamRelease* function is first called to release the compressed stream pointer followed by the video stream pointer. The *AVIFileRelease* does the same function as the *AVIStreamRelase* on an AVI file. The *AVIFileRelease* is called after all the data stream pointers have been freed. Finally, to release the AVIFile library, we call the *AVIFileExit* function. The AVI file is saved at a user specified location and added to the playlist of his/her choice.

In this chapter, we covered the implementation details of how WriteOn1.0 records screen activities and ink in the area enclosed by the scalable eVellum as an AVI file. The functionality to stream a lecture from an instructor machine to student machines over a network is implemented in the Monitor Device component. Chapter 8 details the implementation of Monitor Device component.

# Chapter 8

# Monitor Device

The Monitor Device is primarily a capturing mechanism that solves the problem of capturing the screen activities. The Monitor Device module of WriteOn1.0 implements the capability to capture and broadcast the presentation content as a live compressed video stream from the instructor machine to the student machines. The Remote Capability component, described in Chapter 9, receives, decompresses and renders the lecture stream on the students' computers. The earlier implementation of this module was also responsible for recording a presentation as a movie file on the instructor and student machines. In WriteOn1.0, this functionality has been migrated to the ScreenCapturetoAVI component. Monitor Device is implemented using DirectShow technology.

The Monitor Device in WriteOn1.0 broadcasts the samples of the captured desktop screen activities of the instructor's desktop and his/her ink annotations in real-time to the students in a classroom environment. With the implementation of the functionality to capture the desktop activities as a movie file by the ScreenCapturetoAVI module in WriteOn1.0, the instructor can simultaneously record a local copy of the lecture he/she is presenting to a classroom while broadcasting it. This feature was unavailable in WriteOn because the Monitor Device module facilitated both these functionalities in a mutually exclusive manner. This meant that the user could either capture the screen activities as movie or broadcast it at any point of time but not perform the activities at the same time.

The Monitor Device component provides the WriteOn tool with its core feature by which an instructor had the ability to capture and broadcast a lecture, live, with all the ink annotations made on it by using the eVellum to the student(s). It also provided the capability of preparing and saving lectures composed of computer-based content and ink beforehand by recording them as a movie file, saving the instructor a lot time which could otherwise be consumed making animated PowerPoint presentations. The ability to capture and save and broadcast the instructor

notes as well as the lecture content makes classroom learning and teaching process more effective.

One of the issues concerning the first implementation of the Monitor Device module was its poor operational CPU performance. There are two primary reasons which cause this poor performance:

- the fact that WriteOn can only capture the entire desktop screen area during broadcast and recording
- the use of the WMVideo encoder

WriteOn1.0 solves the first issue by capturing only that area of the desktop over which the user enables the scalable eVellum. This largely reduces the amount of data that needs to be compressed since the scalable eVellum is not usually enabled over the entirety of the desktop screen. The Monitor Device component in WriteOn1.0 modified the CaptureScreenMonitor Filter described in Section 8.2 to include a member function that allows the user to set the capture region. This function was absent in the first implementation of the module because the capture region fixed to that of the entire desktop window. Resizing or relocating the scalable eVellum alters the capture parameters to reflect the new location or dimensions of the same. The MSU Screen Capture codec replaced the WMVideo encoder to compress the frames before broadcasting it because of its better performance statistics as explained in Chapter 5.

The Monitor Device had two implementations in the first version of WriteOn tool. One was using the DirectX's DirectShow technology and the other was using the WindowsMedia Encoder (WMEncoder) technology. Weighing the pros and cons of both the technologies as described in the next section, Section 8.2, DirectX's DirectShow became the technology of choice in WriteOn1.0 and shall remain the technology used for any future enhancements and implementations of the WriteOn tool.

## 8.1 DirectShow vs. WMEncoder

The Monitor Device component in WriteOn1.0 is implemented using DirectX because of its advantages over the WMEncoder technology. Although WMEncoder had advantages over DirectX like better video quality and slightly better operational performance, it had a major disadvantage of not being to capture overlay windows. Since the dynamic content of almost all application windows were visible through the overlay vellum, DirectX had to be technology of

choice because of its ability to capture overlays. DirectShow, based on the COM model has the added advantage of being able to integrate with a lot more devices as compared to WMEncoder. Section 8.2 gives a short overview of DirectX's DirectShow technology.

## 8.2 DirectX's DirectShow® and DirectShow filters

DirectShow® is a multimedia architecture and API that Microsoft® produced as a replacement for the earlier Video for Windows technology. The software developers can use DirectX to perform operations like capture and playback on media files and streams with ease. Based on the Microsoft® Windows® Component Object Model (COM) framework, DirectShow provides a common interface for media operations across programming languages. It supports a wide variety of formats, including Advanced Systems Format (ASF), Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), Motion Picture Experts Group (MPEG) Audio Layer-3 (MP3), and Windows wave (WAV) sound files. A user can create DirectShow components to support new formats or custom effects. Applications like Windows Media Player, Windows Movie Maker, etc. make use of DirectShow to manage and play video and audio media files. With the release of Windows Vista, Microsoft® the Windows Media Foundation (WMF) may replace DirectShow.

DirectShow divides the processing of multimedia tasks such as screen capture, video or audio playback between components known as filters. A filter is a data processing unit that performs some sort of operation on a multimedia stream or file. For example, a filter can perform the task of compressing a video stream. Another example of filter operation is rendering of a video or audio file and so forth. A filter receives input and processes or transforms the input to produce an output. The output of a filter may act as an input to another filter. A DirectShow filter receives input on and mostly produces output on interfaces called pins. An application developed using DirectShow technology performs any task by connecting filters together.

Pins connect one filter to another. Filter pins can be classified as capture pins, input pins and output pins. Capture pins push the samples captured from such devices as webcam, projector, microphone, and so forth; Input pins receive the media samples; and Output pins pass the samples to the input pins of the filter to which they are connected. Each filter is designed to handle a specific type or format of data. For example, a MPEG encoder filter can process only a MPEG stream. Similarly, a MPEG decoder would expect as an input a compressed MPEG

stream. The input pin of one filter and the output pin of the connecting filter must agree on a common media format in order to connect to each other.

A set of connected filters form a filter graph. A filter graph is unidirectional and devoid of loops. DirectShow filter graphs are widely used in video playback (in which each filter will perform a step such as file parsing, video and audio de-multiplexing, decompressing and rendering), video and audio recording, editing etc. . The Filter Graph Manager is a high-level component that controls the filters and provides the methods to construct a filter graph by connecting filters. The Filter Graph manager can start, stop and pause the data flow in a filter graph.

Since DirectShow is based on COM, all the objects and resources created while creating the filter graph must be released without fail before we exit from the application code. The COM based resources are not released automatically during garbage collection and can lead to memory leaks and consumption of large amounts of OS resources.

The following section describes the working of the Capture Monitor Screen filter that is the capture filter of the Monitor Device component. As the name indicates, this filter captures the desktop screen activities and streams to other filters in the filter graph to produce a compressed video stream for broadcasting purposes.

## 8.3 Capture Monitor Screen Filter

The Capture Monitor Screen Filter is the source filter of the Monitor Device filter graph (Figure 8.2). The Capture Monitor Screen filter takes screenshots of the desktop window at a specified rate defined by the frame rate to produce the video stream of the lecture presentation. The frame rate is set to ten frames per second in WriteOn1.0.

The Capture Monitor Screen Filter takes the desktop screen shots using Graphical Design Interface (GDI) operations to capture the bitmap image of the area defined as the capture area and fills the video frame buffer with pixel depth information, dimensions of screen shot and the type of media format. The Capture Monitor Screen filter does not transform the media sample in any manner. At a periodicity specified by the frame rate, the filter graph gives a callback to the Capture Monitor Screen filter to take a screenshot and fill the video frame information such as width, height of the video frame, color of each pixel, the video frame number, etc. The captured

media samples form an uncompressed video stream. A compressor filter compresses the video stream for broadcasting purposes. WriteOn1.0 uses the MSU Screen codec as its compressor filter.

In its first implementation, the capture parameters were hard coded values sized to the dimensions of the complete desktop screen. This meant that the Capture Monitor Screen filter always captured the activities of complete desktop area. This scheme was valid because in the first version of WriteOn the eVellum was enabled over the entirety of the desktop area. However, with the introduction of the scalable eVellum that defines the area of capture, it became necessary to implement a mechanism by which only the portion of the area of the desktop over which the scalable eVellum is captured for broadcasting.

WriteOn1.0's implementation of the filter to realizes the capability that would allow the user to choose an area of his/her choice for screen capture. The scalable eVellum is activated only in the selected area. A function called the SetCaptureArea implemented in the filter code allows the user to set the dimensions of the area he/she wishes to capture for broadcast. The SetCaptureArea function takes as parameters, the screen coordinates of the area selected for capture. Any re-dimensioning of the scalable eVellum changes the parameters of the SetCaptureArea function to reflect the new coordinates of the area of interest.

The Capture Monitor Screen filter is implemented in C++. Since WriteOn1.0 is developed in C#, the filter operates through a COM interface that allows for interoperability between C #and C++ through a COM wrapper in the .Net runtime.

The COM interface of the filter had to be modified in WriteOn1.0 to expose the new SetCaptureArea function. To create the new COM interface, the Interface Definition Language (IDL) file is modified. The .NET IDL compiler compiles the IDL file to generate the interfaces and the Dynamic Link Library (.dll). The Monitor Devices uses the DLL to make the function calls on the capture filter through the exposed COM interface.

## 8.3 Real-time Transport Protocol

The Real-time Transport Protocol (RTP) was designed to transport real-time data like audio and video over the internet. Developed by the Audio-Video Transport Working Group of the IETF, it defines a standard in packet format for transporting audio and video. RTP was originally designed as a multicast protocol for use in multicast conferences involving numerous computers

as the protocol has demonstrated to scale well from two user sessions to sessions with thousands of users. The RTP is usually used in conjunction with the Real-time Transport Control Protocol (RTCP). The RTCP maintains and monitors quality of service (QoS) information and transmission data like packet delays, frame serial numbers, etc. Although documentation on RTP specifies that it be used on Transmission Control Protocol (TCP), it usually uses User Datagram Protocol (UDP). This is because applications using RTP are very sensitive to network latency and lesser sensitive to packet loss.

The main objective of the RTP is to provide services that shall help transport real-time audio and video over IP networks. These services include payload-type identification (indicates what time of content is being transported), packet loss detection and correction, payload and source identification, delivery monitoring, reception quality feedback, media synchronization, and membership management. RTP and RTCP do not provide any guarantee on the QoS or timely packet delivery. They however, do provide the necessary data and information for the application to make local changes to overcome any problem relating to packet delivery. For example, based on the transmission statistics, an application can detect a network congestion and choose reduce the load on network.

The sending application generates RTP frames that contain a payload like a video clip. The RTP frames are loaded onto RTP packets for transmission across the network. An RTP packet can comprise of several RTP frames or one RTP frame may be fragmented among several RTP packets. As seen in the block diagram below, an application buffers the video stream onto its media buffer from which the packets are compressed by a codec before it is transmitted to the network. Each compressed media packet is fragmented into smaller chunks if it is too large and then bundled into RTP packets.

Based on the error correction mechanism implemented at the sending application, the channel coder may generate error correction packets corresponding to an RTP packet before finally sending the packet over the network. The media data remains in the media buffer until the application transmits the RTP packets. It is only after the packets are transmitted that the buffered media data corresponding to the RTP packets are freed. Since the RTP does not implement any error correction scheme, the sender must not discard any data that may be needed for error correction. The sender is also responsible for implementing congestion control mechanism if need be based on the reception quality feedback it receives from the receiving

participants. The sender generates transmission statistics and status reports for the RTP stream it generates.

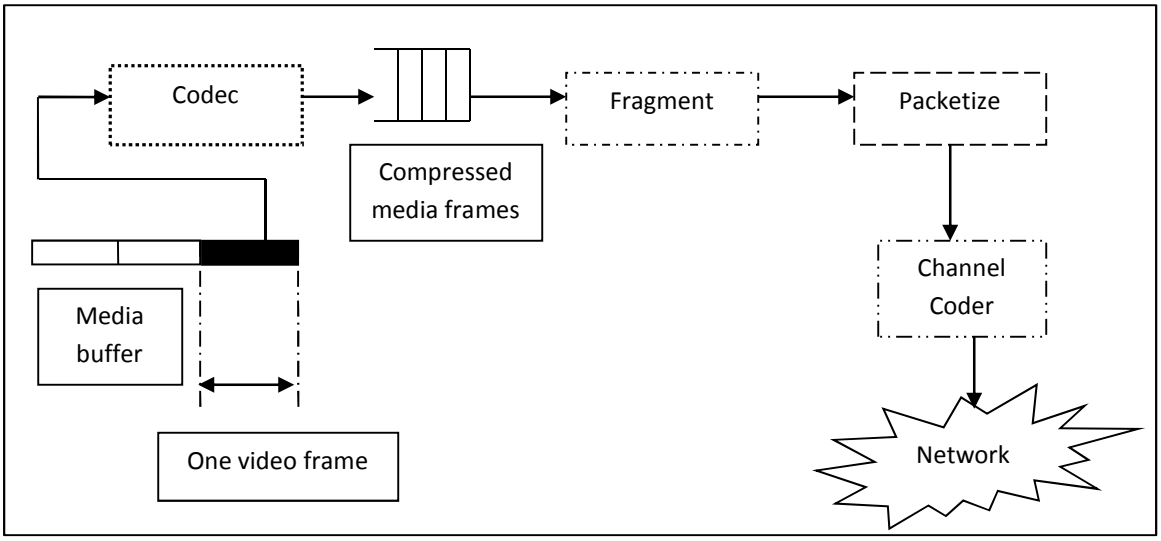



**Figure 8.1: Working of RTP**

The receiving application receives the RTP stream and performs any error correction or recovery of loss packets based on the packet information it receives. The payload is then decompressed from the RTP packet before presenting it to the user. It also sends reception quality feedback to the sending application to help the sender adapt the transmission to the network conditions. ConferenceXP implements RTP API. WriteOn1.0 uses these API's to broadcast a live lecture stream over the network. The following section details how WriteOn1.0 broadcasts activities on the screen of an instructor machine to the students in real-time.

## 8.4 Broadcasting the instructor's presentation

As mentioned before, WriteOn1.0 uses RTP API's of ConferenceXP to implement the screen activity broadcast feature. A series of filter objects work in tandem to capture, compress and broadcast a lecture presentation. Figure 8.2 shows the filter graph of the Monitor Device component of WriteOn1.0 that broadcasts the desktop screen activity of an instructor machine. The Capture Monitor Screen filter in Figure 8.2 takes the screen shots of the desktop and passes the steady stream of screen shots to the MSU Screen codec, which creates a compressed video stream of the lecture.

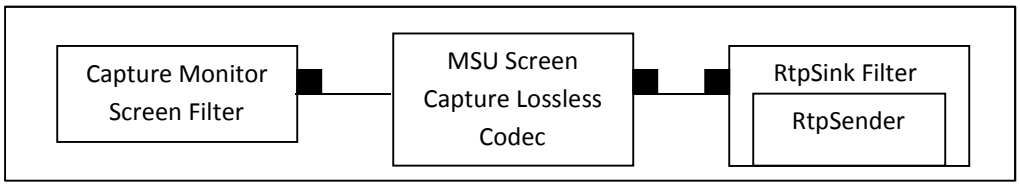



**Figure 8.2: Filter graph used for broadcast of the instructor's desktop screen activity**

Using ConferenceXP API, the instructor and student machines connect to a Conference venue. A venue is a multicast group address used by ConferenceXP to connect a set of computers together over a network. WriteOn1.0 provides a choice two static multicast venues with IP addresses 234.4.4 and 234.5.5.4. The instructor and students first choose the venue before to start a broadcast or receive a lecture respectively. The users connect to a venue using a friendly name of their choice. If any student has the same friendly name as the instructor, he/she cannot receive the lecture.

Technically, the CaptureController module creates an instance of the *RTPSession* class using the friendly name. In WriteOn1.0, a RTP Session comprises of the instructor and the students in a classroom, who are going to communicate using RTP. A participant in an RTP session can simultaneously participate in other RTP sessions. A participant creates an object of *RTPParticipant* class which holds information about the user. The *RTPSession* and *RTPParticipant* objects are also responsible for sending and receiving RTP data, which in our case is the lecture steam. For a participant, a RTP session is identified by a multicast network address, a pair of ports on which data can be sent and a pair of ports on which data can be received. A session may use the same port to send as well as receive data. A port pair comprises of two adjacent port numbers: an even-numbered port for the RTP packets and the next higher odd-numbered port for the RTCP control data.WriteOn1.0 uses the default port pair of 5004 and 5005. Since WriteOn1.0 only transports video payload, we need to create only one RTP session. This is because a RTP session can carry only one type of media.

When the instructor requests that WriteOn1.0 broadcast video samples of the real-time screen activities, the Monitor Device passes the compressed samples to the *RTPSink* filter. The *RTPSink* filter basically serializes the samples and sends them across the multicast session group using *RTPSender*. The *RTPSender* object is created on the instructor machine and is responsible for creating RTP packets and sending the captured data. The MSU Screen Capture codec v1.2

compresses the data before transmission. The *RTPSender* can choose to send data with forward error correction if need be.

The *RTPFilter* pushes the samples into the network. The *RTPSender* object and the *RTPFilter* object are created from the *RTPSession* object. The CaptureController holds a reference to the *RtpSession* object until the user disconnects from the venue. On the student side, an instance of the *RTPListener* class is created to listen for the incoming RTP stream. The *RTPListener* receives the RTP data off the network and extracts the video stream to render it for the students to view.

We have previously described the issues of the poor operational performance of WriteOn during the capture process. Through the implementation of mechanisms like the scalable eVellum and the use of the MSU Screen Capture codec, WriteOn1.0 made significant improvement in the performance of the tool. Research during the development of WriteOn identified another factor affecting the screen capture process. It is the value of the hardware acceleration. The next section summarizes the relation between the value of the hardware acceleration and the capture process.

## 8.5 Hardware Acceleration (HA) and its effect on capture process

Hardware Acceleration refers to the use of the processing power available from the video/graphics card on most Tablet PCs, Notebooks and Desktop PCs instead of the CPU cycles for faster video rendering operations. GDI operations like the line drawing function for example, use the hardware acceleration provided by the graphics card. When hardware acceleration is used during a particular rendering operation, such as drawing a line or playing a video, the video card locks parts of the screen over which it is rendering to create smooth and continuous rendering of a graphic operation. Since the screen capture code is implemented using Windows GDI functions, the capture process will have to wait until the locks in the capture region release, leading to poor performance and delays during the screen capture process. If the hardware acceleration is disabled, the CPU primarily handles the rendering operations, causing no locks and thereby increasing the capture process' operational performance. Thus, for optimum performance during screen capture, the CaptureController of WriteOn1.0 automatically sets the HA of the primary display device, which is the monitor, to a value of five. A value of five for HA in Windows translates to disabling any available hardware acceleration.

Technically, we are setting the registry key "Acceleration.Level" value of the primary display device to level five. The original value of the HA is restored once the user stops the capture process or exits normally from WriteOn1.0. If there is an abnormal termination of the WriteOn1.0 software, the user must restore the appropriate value for HA manually through the Windows display properties dialog window.

The Monitor Device component streams a compressed video of the lecture material and ink over the network for the Remote Capability component to receive, decompress and render the video stream on the student machine. The next chapter, Chapter 9, details the Remote Capability component of WriteOn1.0

# Chapter 9

# Remote Capability

The Remote Capability component in WriteOn1.0 is responsible for receiving the lecture presentation broadcast by the instructor and rendering it on the student desktop screen. In order to start viewing the live lecture presentation on his/her desktop, the student connects to the same virtual classroom to which the instructor is connected. The virtual classroom is technically a multicast venue to which the instructor broadcasts the presentation material. Functions exposed by the remote capability component facilitate the student(s) to connect to the virtual classroom. Remote capability creates a new window rendering the video stream on the student desktop as soon as it receives the first frame of the broadcast video stream at the virtual classroom venue.

In order for students to make their own notes, they can enable the scalable eVellum from the WriteOn1.0 toolbar on the area of their window rendering the video stream. This feature of allowing the student to enable the scalable eVellum over the lecture window at a point of time of his/her choice was absent in the earlier version of the tool. Instead, the window rendering the video stream was inlaid with the eVellum. Although enabling of the eVellum by default was a very convenient for taking notes, the students lacked the flexibility of changing the ink color, pen width, as well as the area over which they would like to annotate. The scalable eVellum of WriteOn1.0 addresses these issues to enhance the note-taking process.

The students can save a copy of the lecture session on their computer as a movie playlist. The ScreenCapturetoAVI module of WriteOn1.0 facilitates the student to record and save the classroom notes that include instructor's annotations along with the notes he/she takes as a movie playlist. This recorded playlist of a lecture session constitutes the notes the student may use for later review and study.

Remote Capability uses DirectShow technology for rendering the video stream broadcast. The following section gives an overview of the technical details about how DirectShow renders a video stream in WriteOn1.0.

## 9.1 Rendering lecture on student(s) desktop

Remote Capability uses DirectShow technology of DirectX to render the video on the student desktop. The remote capability component implements a RTP stream listener that that listens for the first RTP video packet. As soon an as the first frame of the incoming RTP stream is received, the remote capability component creates the DirectShow filter graph for rendering the video. The RemoteDesktopController class of WriteOn1.0 implements the instantiation of the remote capability component. In order to view the live lecture stream a student connects to the same multicast venue as the instructor and joins the RTPSession initiated by the instructor. The students can choose a name of their choice using which they would like to connect. However, if a student connects to the RTPSession with the same name as another student or the instructor, they cannot receive the lecture. An instance of the RTPStream class is created for the incoming lecture stream.

Figure 9.1 shows the filter graph used by the RemoteCapability class. The RTPSource filter (a ConferenceXP RTP filter) in the figure is initialized with the RTPStream object. The RTPSource filter reads the video samples from the RTPStream. The RemoteCapability builds the filter graph upon receiving the first video sample, because it has to know the media format details such as width and height of the video image, the bit rate, frames per sec and other information. Since the RTPStream is in a compressed state, a video decoder such as the WMVideo decoder is connected to the RTPSource. The decompressed RTP video stream is rendered on the student computer on a separate window.

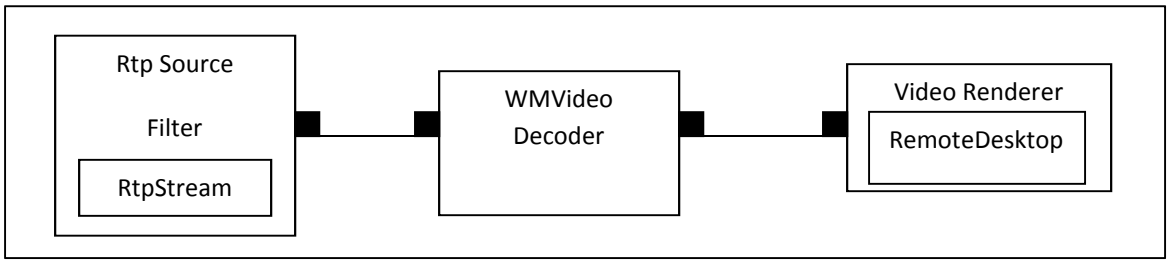



**Figure 9.1: RemoteCapability filter graph**

The RemoteDesktop class of the remote capability component implements the windows form on which the video of the instructor's presentation is rendered on the student computer. The RemoteDesktop class instantiates an instance of the video renderer filter. The video renderer

filter is attached to the video decoder. The video renderer receives the decompressed RTP video frames and displays it on the RemoteDesktop window.

The next section of this chapter explains how the students can perform note-taking using WriteOn1.0.

## 9.2 Annotating on RemoteDesktop window

Remote Capability component creates an instance of the RemoteDesktop form in a new window to render the video on the student desktop. In the earlier version of the tool, the RemoteDesktop window had the virtually transparent eVellum enabled in its area for student note-taking. The eVellum inlaid in the window rendering the lecture provided the students the ability to take notes on the lecture material itself while allowing them to remain in the context of the lecture while performing note-taking. However, the students could not change the pen color and width using this scheme of note-taking. The students also lacked the flexibility of changing the size of the eVellum as per their needs.

In order to allow the students to perform easy note-taking, WriteOn1.0 allows them to perform note-taking using the scalable eVellum. The inlaid eVellum of the RemoteDesktop window is disabled in WriteOn1.0. The student can enable the eVellum from the WriteOn1.0 toolbar. The student can choose to enable the eVellum only on the window rendering the video to make his/her notes over the incoming lecture stream. The students can select the pen width and color from the WriteOn1.0 toolbar. If the student wishes to start recording the lecture for review at a later point of time, he/she can do so by choosing the appropriate menu item from the WriteOn1.0 toolbar. The student notes would be the composite of the lecture material from the instructor, the instructor notes as well as his/her own notes. With WriteOn1.0, the student can perform better note-taking because of the flexibility and ease of use offered by the scalable eVellum. The student saves his/her notes as movie playlist on the local computer using the ScreenCapturetoAVI module.

Until now, we covered the technical details of the implementation of WriteOn1.0 and its features. The following chapter presents the results and potential future work on this application.

# Chapter 10

# Results, Conclusions and Future Work

The first section of the chapter presents a comparative analysis of the application performance statistics in terms of CPU resource consumption of both the versions of WriteOn. Section 10.2 which explains the key conclusions of the research efforts of this thesis follows the first section. The final section of this chapter gives recommendations for future research on WriteOn1.0.

## 10.1 Results

In this section of the chapter, we shall present some important results from the pilot test of using WriteOn1.0. We shall see how the new implementation of the tool fared in its operational CPU performance during a screen capture session. We shall also evaluate the user experience of using the new GUI and the scalable eVellum. The first subsection presents the results of the CPU performance of WriteOn1.0.

### 10.1.1 Operational CPU performance

One of the highlights of WriteOn1.0 is its operational CPU performance. WriteOn1.0 has significantly optimized the CPU resource utilization during a screen capture session. The WriteOn1.0 tool has been tested to assess its operational CPU performance statistics and a comparative analysis of the application's performance was done as compared to its previous version.

WriteOn and WriteOn1.0 were installed on Toshiba tablet PC's with the identical hardware and software configurations. Both the tablet PC's had the same solid black wallpaper. Identical LogicWorks® simulations were initiated on both the tablet PC's. The screen capture process was initiated with the same screen dimensions for both versions of the tool, beginning with an area of 200X300 pixels. The screen capture process was allowed to run over a span of

five minutes and the average CPU resource utilization was noted. The Table 10.1 summarizes the results of the load tests.

| Screen Capture Dimension | Average WriteOn CPU usage % | Average WriteOn1.0 CPU usage % |
|---|---|---|
| 200 * 300 | 60 | 20 |
| 400 * 600 | 80 | 40 |
| 1024 * 768 | 90 | 60 |

**Table 10.1: CPU performance statistics of WriteOn and WriteOn1.0**

As seen from the results, WriteOn1.0 showed a marked improvement in its operational CPU performance. WriteOn1.0, with the use of the MSU Screen Capture codec v1.2 improved the operational performance by at least 30 percent. Unlike WriteOn, there were no signs of the system stalling due to a resource crunch in while capturing the entire desktop area of 1024 x 768.

### 10.1.2 Size of AVI file

While recording a lecture session to a local AVI file, a user would like the AVI file size to be optimum because of the fixed availability of disk space. WriteOn and WriteOn1.0 were compared for the sizes of the AVI files they generated. The screen capture area was set to 1024 x 768 and the screen activities in the area of capture were recorded to an AVI file for 60 seconds.

The AVI files were then compared to note the size differences between them. The sizes of the AVI files generated were observed to be an order of magnitude smaller with the use of the MSU screen codec v1.2. as in WriteOn1.0 when compared to WriteOn, that generated AVI files using the WMVideo 9 codec. The table below summarizes the results of this test.

| Area of capture | Codec | AVI file size (1 min movie) |
|---|---|---|
| Full Screen (1024 x 768) | WMVideo9 | 3.50 MB |
| Full Screen | MSU screen capture v1.2 | 2.27 MB |

**Table 10.2: Sizes of AVI file generated by WriteOn and WriteOn1.0**

### 10.1.3 User experience of using WriteOn1.0

WriteOn1.0 received positive feedback in terms of its GUI and the overall user experience of using the tool. The flexibility given by scalable eVellum received very positive reviews from an instructor. The transparent mode of operation of the scalable eVellum that provided the capability of accessing the desktop without having to toggle the scalable eVellum was well appreciated. The ability of the user to perform other functionality in WriteOn1.0 with ease from the toolbar with a single click was welcome by the users. At the student end, there was a reduction in the lag while receiving the lecture. Due to the implementation of the WriteOn1.0 toolbar, the lag encountered in enabling the tray menu of WriteOn is eliminated. The next section of this chapter summarizes the key conclusions of this research project.

## 10.2 Conclusions

The primary aim of the research efforts in this thesis project was to successfully implement the second core version of WriteOn, WriteOn1.0, which shall be an improved functional version of the tool. WriteOn1.0 was successfully built and deployed to evaluate the new features of the tool and its operational performance. Primary tests have shown satisfactory results with respect to the new UI. As discussed in the results, WriteOn1.0 showed a marked improvement in its performance issues with respect to the CPU resource usage. The feature to record in parallel, a live lecture that is being broadcasted using WriteOn1.0 is expected to give the instructor the ability to review the lecture he/she presented in the classroom and improve upon it before making it available to the students. The transparent mode of operation in the scalable eVellum enables the user to access any control buttons of an application like a simulator's start and stop button without having to toggle the eVellum. The instructor(s) and student(s) shall appreciate the transparent mode of operation of the scalable eVellum because he/she can remain in context, view all the ink and access the underlying application(s), all the same time. The instructor experience of using the WriteOn1.0 is enhanced by the improvement in application performance during screen capture process. The ability to move the eVellum to a location of his/her choice is expected to allow him/her to focus attention of students to specific

areas/topics of the lecture instantly. The new UI eliminates the need to fill in fields in a dialog in order to set up various parameters for the screen capture and broadcast process.

Although the second implementation of WriteOn is a significant improvement over the early version of the tool, there are several aspects of the application that can be developed to make it more comprehensive and complete. The next section presents the possible features that can be implemented as part of future work on WriteOn1.0.

## 10.3 Future Work

WriteOn1.0 can be deployed in all classroom environments as an effective lecturing tool because of its features like the virtual transparency or the scalable eVellum over which the user can annotate using electronic ink, live streaming of both static and dynamic lecture material to the students and the ability to make notes on an active lecture stream without losing its visibility. However, there is scope for implementation of many additional features in WriteOn1.0 to make it more comprehensive teaching tool. One of the key feature additions to WriteOn1.0 is the ability to perform keyword searches in existing lecture movie files. This search feature can be implemented using handwriting recognition API's of the .Net framework to recognize the ink annotations made by the user and save the results in a text file. A separate text file containing the text of all the ink annotations can be associated with every lecture movie file. Thus, when the user performs a search operation from the WriteOn1.0 tool, all text files associated with the lecture playlist can be searched and the appropriate playlist can be returned.

Another key area for future work on this tool can be that of student feedback and student submissions. Real-time feedback and submissions have the potential to enhance the active learning process in classrooms. The support to broadcast the instructor's voice along with the lecture content would be another valuable feature to WriteOn1.0. In the area of application development, an update of the application development framework to the latest version of the .Net runtime is an important task. The code base has to be modified to use the latest version of the ConferenceXP API's. The above-mentioned features shall greatly enhance the current version of WriteOn1.0 tool to make it a more complete and comprehensive application for teaching and learning.

# References


1. Anderson, Richard J., "Classroom Presenter", website, *http:/ /www.cs.washington .edu/education/dl/presenter/*, Computer Science & Engineering, University of Washington website.

2. Joseph G. Tront, Vinod Eligeti and Jane Prey, "Classroom Presentations Using Tablet PCs and WriteOn", 36th ASEE/IEEE Frontiers in Education Conference, October 28 – 31, 2006, San Diego, CA

3. Intons-Peterson, M. J. & Fournier, J. (1986), " External and internal memory aids: When and how often do we use them? ", Journal of Experimental Psychology: General, 115, 267-280.

4. Rune Kornefors and Lennart Lundberg, "Computer-Aided Teaching in Teacher Training", Department of Engineering and Natural Sciences, Viixj6 University, S-35 1 95 Vhjo, Sweden, 12/11/1996

5. Microsoft's OneNote® 2003 product information, *http://www.microsoft.com/office/onenote/prodinfo/default.mspx*

6. Microsoft Windows® Journal®, *http://www.microsoft.com/windowsxp/tabletpc/evaluation/overviews/pctools.mspx#EYC*

7. Richard Anderson, Ruth Anderson, Crystal Hoyer, Beth Simon, Steve A.Wolfman , Tammy VanDeGrift and Ken Yasuhara, "Experiences with a tablet PC based lecture presentation system in computer science courses". Proceedings of the 35th SIGCSE technical symposium on Computer science education, 2004

8. Michelle Wilkerson, William G. Griswold and Beth Simon, "Ubiquitous Presenter: Increasing Student Access and Control in a Digital Lecturing Environment", ACM Special Interest Group on Computer Science Education (SIGCSE) 2005

9. Tront, Joseph G., "Using Tablet PCs in Engineering Education", 2005 ASEE Annual Conference & Exposition, Portland, OR, June 18-21, 2005.

10. Joseph G. Tront and Jane Prey, "WriteOn: A Tool to Support Teaching Software Engineering" ,Proceedings of the 19th Conference on Software Engineering Education and Training Workshops (CSEETW'06)

11. Microsoft® Remote Desktop - *http://support.microsoft.com/kb/284931*

12. Moscow State University Graphics and Media Lab, developers of MSU Lossless Screen Capture Codec- *http://compression.ru/video/ls-codec/screen_capture_codec_en.htm*

13. Official website of DyKnow corporation, *http://dyknow.com*

14. Camtasia Studio, developed by TechSmith Corporation, *http://www.techsmith.com/camtasia.asp*

15. CamStudio *http://camstudio.org/*

16. Richard Anderson, Ruth Anderson, Crystal Hoyer, Beth Simon, Fred Videon, and Steve Wolfman , "Lecture Presentation from the Tablet PC". Workshop on Advanced Collaborative Environments, Seattle, Washington, 2003.

17. Richard Anderson, Ruth Anderson, Crystal Hoyer and Steve Wolfman**. "**A Study of Digital Ink in Lecture Presentation", Conference on Human Factors in Computing Systems (CHI), 2004, pp. 567-574, April, 2004.

18. Beth Simon, Ruth Anderson, Crystal Hoyer and Jonathan Su , "Preliminary Experiences with a Tablet PC Based System to Support Active Learning in Computer Science Courses", Innovation and Technology in Computer Science Education (ITICSE), 2004

19. Dave Berque, "Promoting Classroom Interactivity in Computer Sciences Courses using Laptops , Pen-Based computers, Tablet PC's and DyKnow Software".

20. Rockman Et Al, "A case study of DyKnow Vision: Conversations and observations that demonstrate its educational potential.", Jan 2007

21. Hrepic, Z, "Utilizing DyKnow Software and Pen-Based, Wireless Computing in Teaching Introductory Modern Physics", Fort Hays State University , 2006.

22. Scott Turner, Kibum Kim, Manuel A. Pérez-Quiñones, Stephen H. Edwards, "Note taking and the Tablet PC", 2006

23. Real Virtual Network Computing, Remote access software and mobile device access, *http://vnc.com*

24. Smith, L. and Smith, E , "Using Camtasia to develop and enhance online learning", University of Louisiana, Monroe

25. Wassgren, C. and Krousgrill, C., "Pen-Based PCs for the Classroom, Teaching and Learning with Technology Workshop" , 2003 *http://widget.ecn.purdue.edu/~wassgren/2003_04_03_TLTWorkshop*

26. Hartsell, T., & Yuen, S. (2006). "Video streaming in online learning". AACE Journal, 14(1), 31-43.

27. Jaspal Subhlok, Olin Johnson, Venkat Subramaniam, Ricardo Vilalta, and Chang Yun, "Experience with Tablet PC Video based Hybrid Coursework in Computer Science", University of Houston, 2007

28. Model View Controller Paradigm, *heim.ifi.uio.no/~trygver/mvc/index.html*

29. NET Framework, Microsoft Corporation, *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpconNETFrameworkTools.asp*

30. Microsoft Visual Studio *http://www.microsoft.com/visualstudio/en-us*

31. Video for Windows®, *http://msdn.microsoft.com/en-us/library/ms713492(VS.85).aspx*

32. Conference XP Research Project, *http://research.microsoft.com/conferencexp/*

33. Object-Oriented Paradigm by Peter Coad , Jill Nicola (ISBN-10: 013032616X | ISBN-13: 978-0130326164)

# Appendix I

## WriteOn1.0 User's Manual

[Co-Authors: Dr. Joseph G Tront, Vinod Eligeti]

This appendix is the User Manual for WriteOn1.0 providing technical details for using the software in a classroom. WriteOn1.0 finds its use as a tool to support the instructor in a lecture session and as an effective note-taking tool for the students. There can only be one instructor in any lecture session. This appendix illustrates, in a step-by-step fashion, how each of the two user groups can effectively use the tool in the classroom.

Before going into the details of how to use all the features of WriteOn1.0, we shall present a "Quick Start" manual for both the instructor and student user groups. The "Quick Start" description for the instructor group shall give the steps on how to initiate a broadcast session and annotate on lecture presentation. For the student group, the "Quick Start" description presents the steps necessary to initiate a receive session, save the lecture on the local disk and perform note taking.

After presenting the Quick Start tutorials, we shall present a detailed version of the WriteOn1.0 tutorial.

Table of Contents

## I. Installing WriteOn1.0

i. Download the WriteOn1.0 installer from *http://filebox.ece.vt.edu/~jgtront/tabletpc/writeon.html* by clicking on the "download" menu option from the menu items on the left-hand side of the webpage as seen in Figure 1.

ii. Upon clicking the "download" menu option, the pop up dialog for saving the Windows Installer Package for WriteOn1.0 appears. Click the "SaveFile" option to download the installer file on to your desktop or a folder of your choice. Figure 1 also shows the "Save WriteOn.msi" popup dialog.

iii. Once the download is complete, locate the WriteOn_Setup.msi file on your computer and double-click it to begin the installation process. Figure 2 shows the WriteOn.msi file located on the user's desktop. Please note that you have to uninstall any previous versions

of WriteOn (Uninstall WriteOn) already installed on your computer for successful installation of WriteOn1.0.

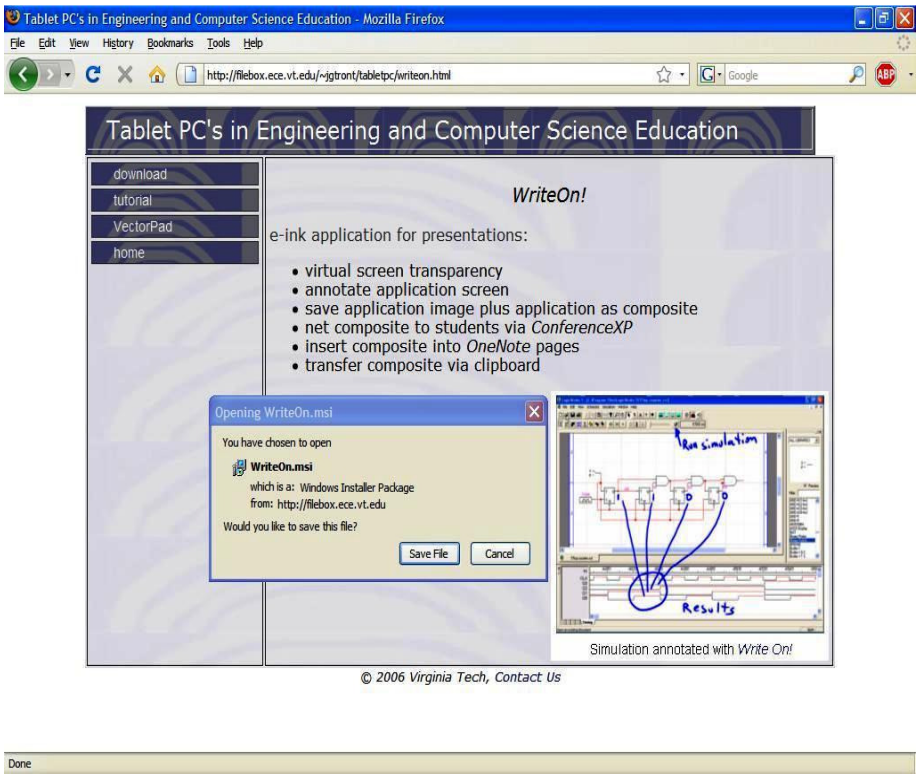



**Figure 1: Download WriteOn1.0 Installer**

iv. Upon double-clicking the WriteOn.msi file, the WriteOn1.0 setup wizard dialog appears as seen in Figure 3. Click on the "Next" button from the bottom right-hand corner of the dialog to continue with the installation process.


**Figure 2: WriteOn1.0 Installer on User's Desktop**


**Figure 3: WriteOn Setup Wizard Dialog**

v. The next step is to choose the location where in you would like to install the WriteOn1.0 files. The default location is *C:\Program Files\ Virginia Tech\WriteOn* . It is highly recommended to install WriteOn1.0 at this default location. Figure 4 shows a snapshot of the select installation folder dialog.

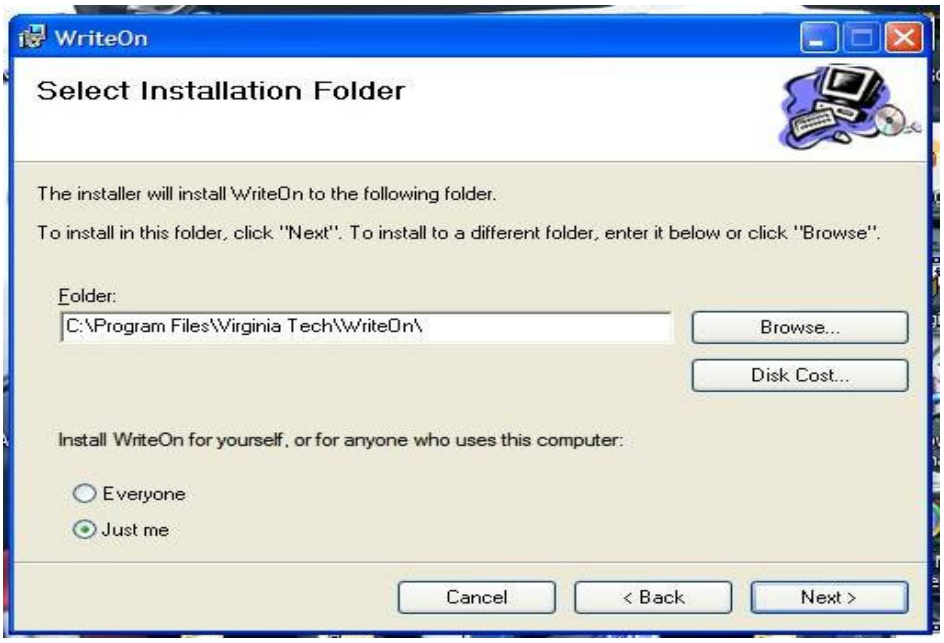



**Figure 4: Choose Location for WriteOn1.0 installation**

vi. After you have chosen a location to install WriteOn1.0, click on the “Next” button from the bottom right-hand corner of the dialog to confirm that you would like to install WriteOn1.0 as seen in Figure 5. If you have confirmed to install WriteOn1.0, the installation begins and Figure 6 shows a screen shot of the progress of the installation process.

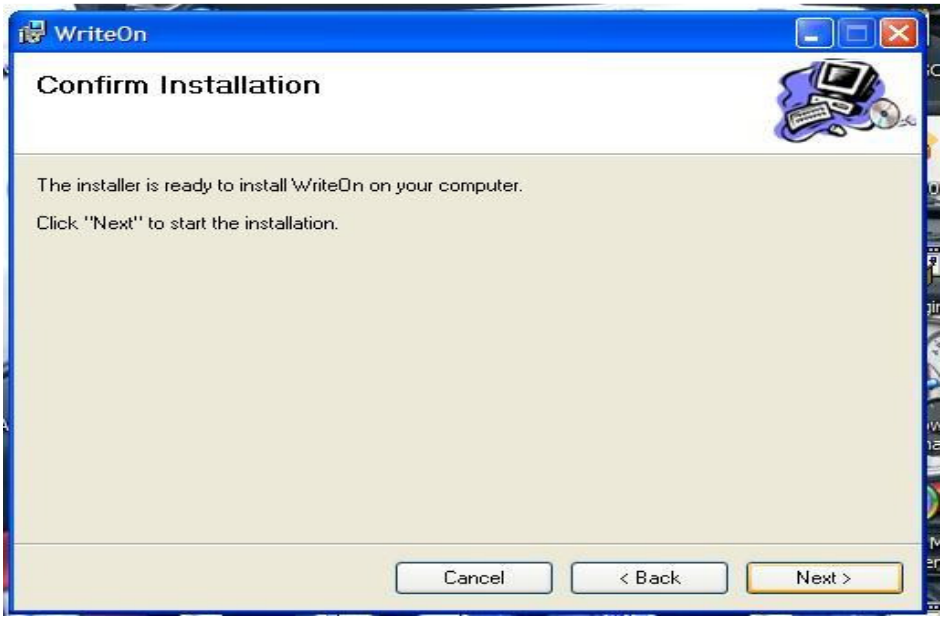



**Figure 5: Confirm WriteOn1.0 installation**

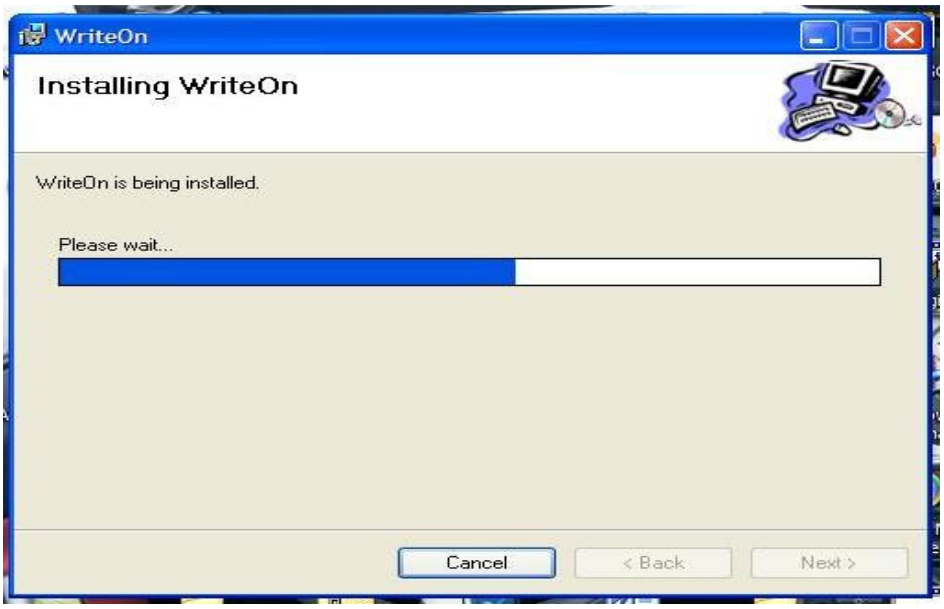



**Figure 6: Progress of WriteOn1.0 installation**

vii. During the installation process, the installer installs the MSU screen capture codec v1.2. Please read the user agreement and then click the “Yes” button at the bottom of the dialog as seen in Figure 7. Figure 8 shows the snapshot of the installer wizard dialog after the MSU codec has been successfully installed.

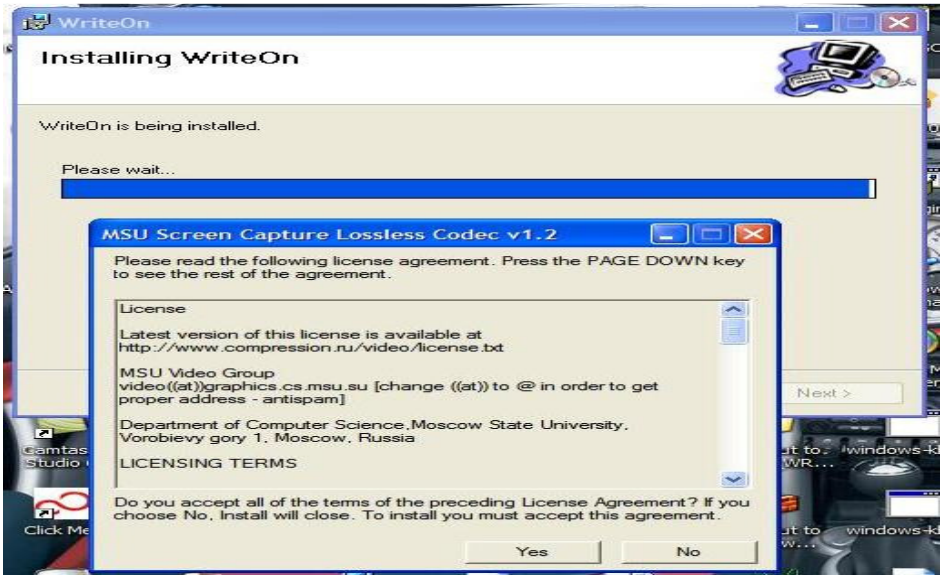



**Figure 7: Installing MSU Screen Capture Lossless Codec v1.2**


**Figure 8: MSU Screen Capture Lossless Codec v 1.2 installed**

viii. Figure 9 shows the snapshot of the installer dialog after WriteOn1.0 has been successfully installed. The WriteOn1.0 icon appears on the user's desktop as seen in Figure 10.

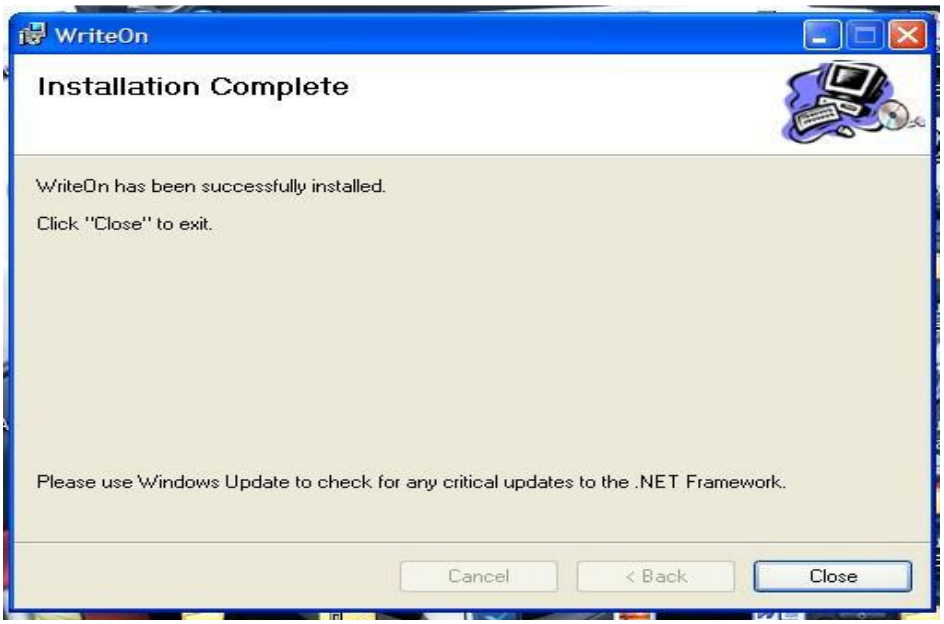



**Figure 9: WriteOn1.0 Installation Complete**

## II. WriteOn1.0 Quick Start for Instructors

### 1. Getting Started with WriteOn1.0

Ensure that you have installed WriteOn1.0 (see section Installing WriteOn1.0) on your computer. In order to begin using WriteOn1.0, the instructor starts the application by double clicking on the application's icon on the Windows desktop as seen in Figure 10. Once WriteOn1.0 is loaded, the selection window of Figure 11 is displayed.


**Figure 10: WriteOn1.0 icon**

**2. Initiate a Broadcast session**

A typical scenario is one in which the instructor uses WriteOn1.0 to broadcast a lecture presentation over a network to the classroom of students. In order to initiate a broadcast session, the instructor chooses the "Broadcast a Lecture" menu item from the WriteOn1.0 Main Form as shown in Figure 11.


**Figure 11: WriteOn1.0 Main Form**

The instructor must then choose a venue to connect to by clicking on the appropriate radio button from the broadcast lecture form as seen in Figure 12. There are two multicast venues supported in WriteOn1.0 with addresses 234.4.4.4 and 234.5.4.4, respectively.

Once the instructor has connected to a multicast venue/classroom, the next step is to select an area of the screen to broadcast from the "Select Region" dialog. The "select region" form appears as shown in Figure 13. The "Region" radio button allows the instructor to choose an area of his/her choice for broadcast while selecting the "Full Screen" radio button broadcasts activities on the entire screen.

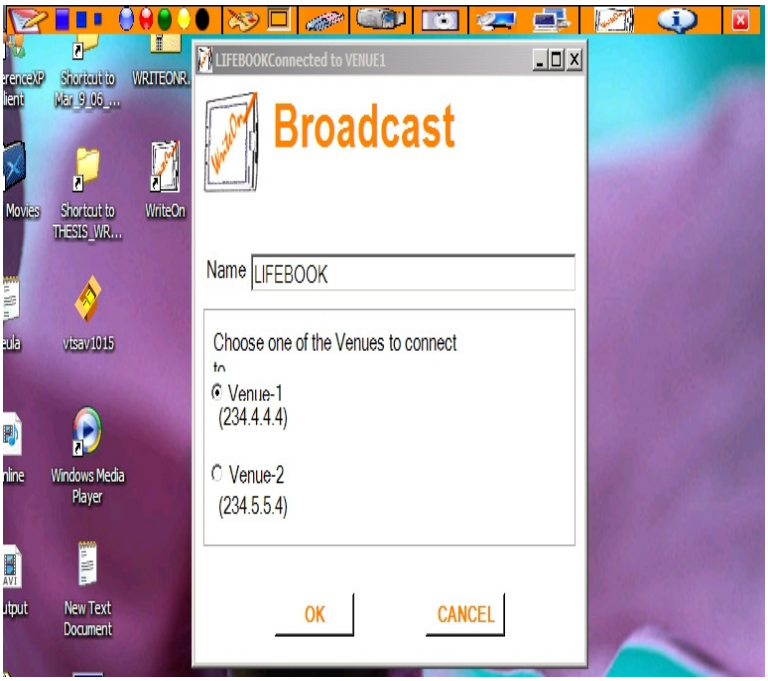



**Figure 12: Instructor choosing Venue-1 from the broadcast lecture form**

If the user has selected the "Region" option, a selection cursor appears on the users screen. In order to define an area for broadcast, the user can click down on the start coordinates and then stretch the red dotted border to encompass the area of his/her choice without lifting the mouse button. Figure 14 shows the user selecting an area for broadcast.

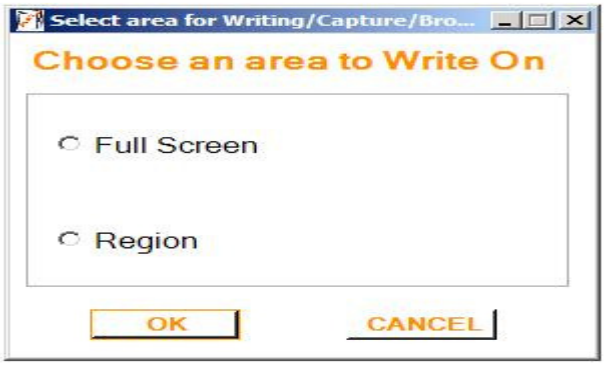



**Figure 13: Form to select area of interest for capture/writing**


**Figure 14: Selecting area on screen**

After the area is chosen, the desktop activities in it begin streaming to the student computers connected to the same classroom/venue. Following these initial set up steps, the WriteOn1.0 toolbar becomes visible in the middle of the top edge of the Tablet PC screen. The "Start Broadcast" icon on the toolbar is highlighted to green to indicate that a lecture is being broadcast as seen in the Figure 15.


**Figure 15: Lecture broadcast in progress**

### 3. Start Writing

At this point, the instructor can use WriteOn1.0 for annotating by activating the scalable eVellum by clicking on the "Start Writing" icon on the tool bar as shown in Figure 16. The user should note that rolling the mouse pointer over a menu button on the Writeon1.0 tool bar will cause the name/function of the button to be displayed.


**Figure 16: Start writing icon on WriteOn1.0 toolbar**

If the user has chosen to broadcast the activities of the entire screen in the previous step, then the scalable eVellum is deployed over the entire screen. If the instructor opted for the

"Region" option in the previous step, then the eVellum is deployed only in that restricted area as seen in Figure 17. Activities only the screen area selected will be broadcast.


**Figure 17: Annotation on scalable eVellum enabled over a region**

With the Start Writing button selected as shown in Figure 17, the instructor may begin annotating and produce electronic ink within the red borders of the scalable eVellum as seen in the figure above. The default ink color is orange.

The instructor can chose to change the area over which he/she is writing by resizing the scalable eVellum. The scalable eVellum can be resized by placing the mouse cursor on its border and then clicking down and dragging the border to the desired location. The mechanism of resizing the scalable eVellum is similar to that of any window on the user desktop.

## III. WriteOn1.0 Quick Start for Students

### 1. Getting Started with WriteOn1.0

Begin using WriteOn1.0, by double clicking on the WriteOn icon on the Windows desktop as shown in Figure 10.

### 2. Initiate a Receive Lecture session

To receive a broadcast session, the student must first connect to the appropriate venue/classroom using the "Receive Lecture" form that appears when the user clicks on the

"Receive a Broadcasted Lecture" button from the WriteOn1.0 Main Form. The WriteOn1.0 Main form appears when the program initiates and is shown in Figure 11. The specific "Receive a Broadcasted Lecture" button is shown in Figure 18.


**Figure 18: Receive lecture button on WriteOn1.0 Main Form**

A "Receive Broadcast" form will appear on the student machine as shown in Figure 19. The student can change the friendly name by which he/she would like to connect to the classroom by editing the text in the "Name" textbox. A connection is established once the student chooses a "Venue" by clicking on the appropriate radio button and clicks OK.

Once the student is connected to the venue, the "Receive a Lecture" icon on the WriteOn1.0 toolbar is highlighted to indicate the receipt of a lecture and the live lecture stream is rendered in a separate window as shown in the screenshot in Figure 20.

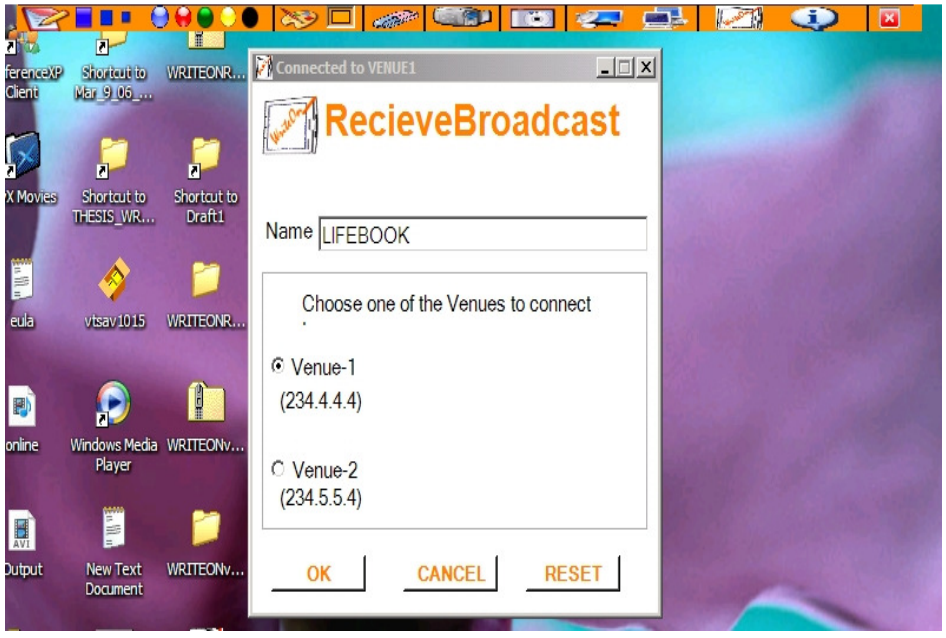



**Figure 19: Student choosing venue from the receive lecture from**

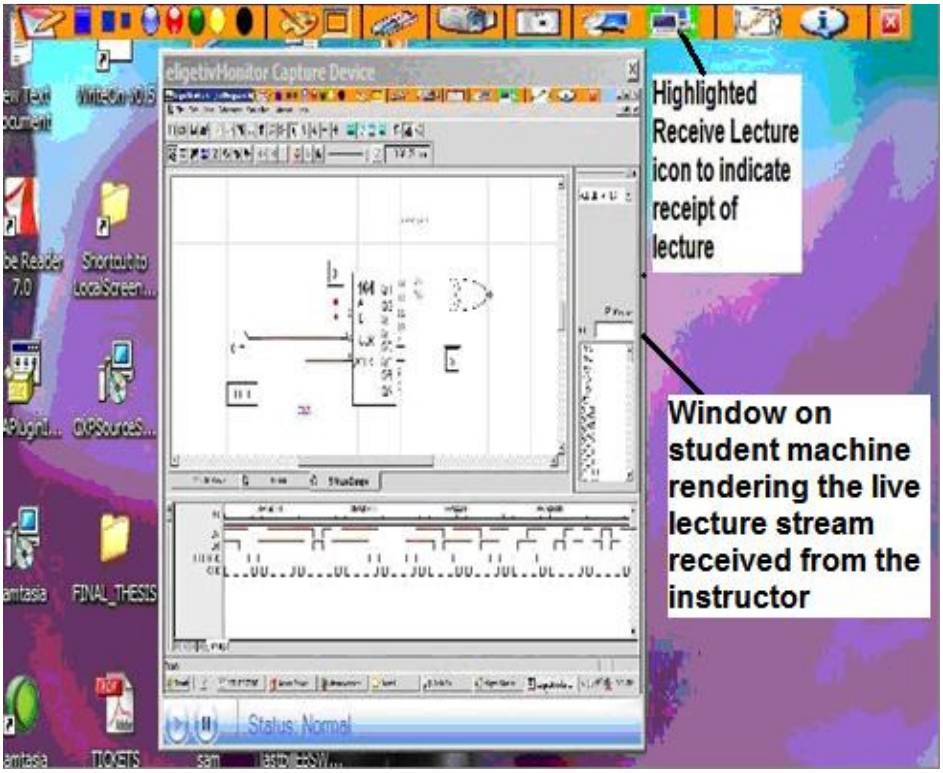



**Figure 20: Receive lecture in progress**

### 3. Recording a Lecture to a Movie file on the Local Disk

To begin saving material broadcast by the instructor as a movie file, the student must click on the "Capture Screen Activity as Movie Playlist" icon on the WriteOn1.0 toolbar as seen in Figure 21.


**Figure 21: Capture Screen activity as Movie Playlist icon**

The student must then choose an area of interest to record as a movie. The "Select Region" form appears as shown in Figure 22 below.

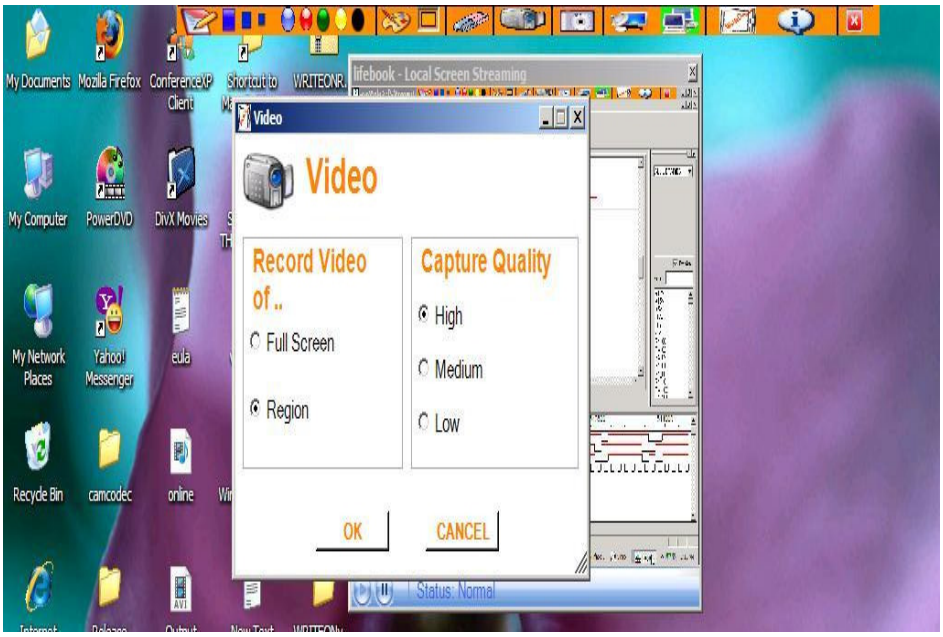



**Figure 22: Student choosing area and quality of the lecture movie**

If the user has chosen the "Region" option, the select cursor appears on the screen. In order to define an area for recording, the user can click down on the start coordinates and then stretch the red dotted border to encompass the area of his/her choice without lifting the mouse button (see Figure 14). The point at which the user lifts the mouse button is the end coordinates of the area for recording.

Once the area and quality are chosen, the student is asked to choose a name by which he/she is going to save the video recording. Recordings are saved as movie playlists and the student can navigate to and save in any directory of their choice. The default directory is Lectures on the local home directory. The screenshot in Figure 23 shows the form used to save the recording with the name "lecture1" in the "Lectures" folder.

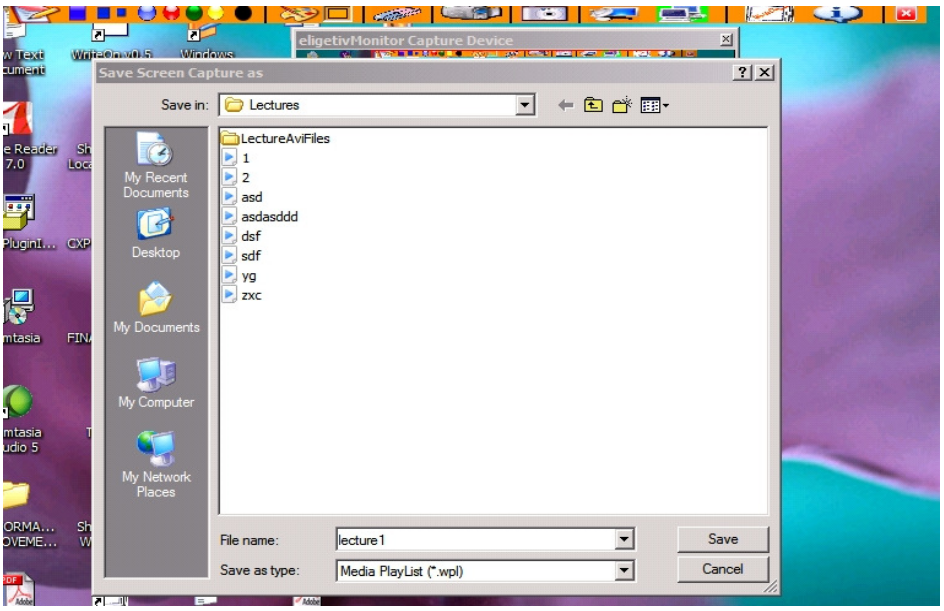



**Figure 23: Student choosing name of lecture file to record lecture**

After a name has been chosen, recording begins and the "Capture Screen Activity as Movie Playlist" icon on the toolbar is highlighted as seen in the screenshot below in Figure 24.

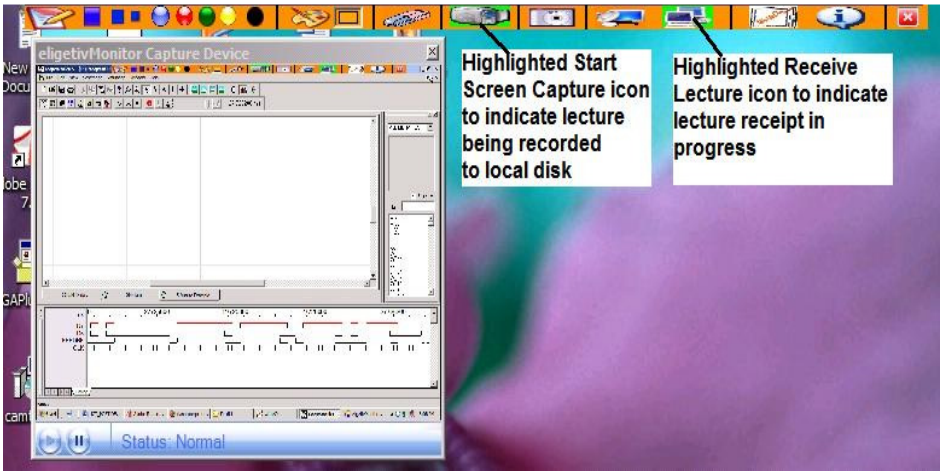



**Figure 24: Student receiving and recording lecture**

## 4. Start Note-taking on the received Lecture

The student can use WriteOn1.0 to annotate on the received lecture material by activating the scalable eVellum over the window displaying the lecture from the instructor. Simply click on the "Start Writing" icon on the tool bar as was shown in Figure 16. If the student is recording the activities in a particular region on the screen, the scalable eVellum is placed over the same

region. Figure 25 gives a screenshot of a student taking notes on the lecture material and recording it.

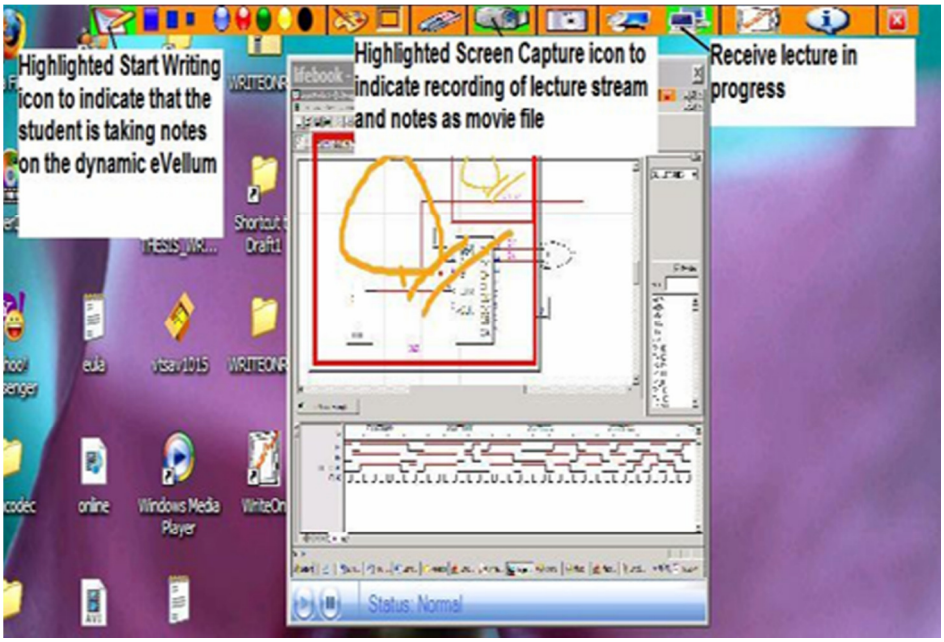



**Figure 25: Student performing note taking on lecture stream**

## IV. WriteOn1.0 Tutorial

This section of the appendix is a tutorial that provides detailed information on the operation of WriteOn1.0 that goes into more depth than the Quick Start manual presented above.

### 1. Start WriteOn1.0

The user can launch WriteOn from the Start Menu by choosing:

Start → All Programs → Virginia Tech → WriteOn.

Figure 26 shows graphically how to launch WriteOn1.0 from the Start menu

The Main Form of WriteOn1.0 appears shortly after the application starts up as shown in the screenshot in Figure 11.

The Main Form contains five menu options namely, “Start Writing”, “Capture Screen Activity as Video”, “Broadcast a Lecture”, “Receive a Broadcasted Lecture” and “Take me to WriteOn Tool Bar”. The first four options indicate the primary tasks supported in WriteOn1.0. We shall talk about each of the first four options in the coming sections of this appendix. The fifth option takes the user directly to the WriteOn1.0 tool bar.

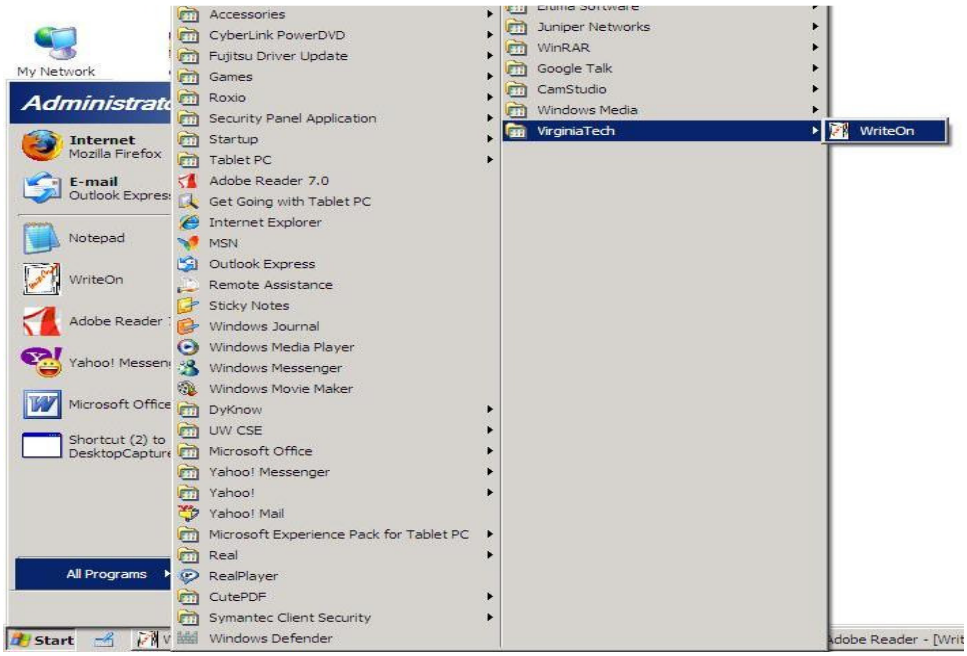



**Figure 26: Choosing WriteOn1.0 from Start menu**

The WriteOn1.0 tool bar is a form which docked to the upper edge of the user’s Windows desktop. In addition to the options provided in the Main Form, the tool bar provides a variety of supplemental functions such as pen size and color modification, snapshot capture, and ink erasure.

The toolbar is picture based i.e., the toolbar is comprised only of icons. When the mouse cursor hovers over an icon, tool tips appear which explain the actions performed when the icon is clicked. Figure 27 below is a screenshot of the WriteOn1.0 toolbar when we choose the “Take me to WriteOn toolbar” option from the Main Form. The user can choose to perform tasks like broadcasting a lecture or receiving a lecture from the Main Form or the toolbar.

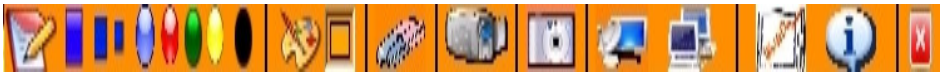


**Figure 27: WriteOn1.0 toolbar**

The next section explains in detail how both the instructor and student can use WriteOn1.0 for performing common tasks like activating the vellum for ink annotations, changing the ink color, etc. In addition to the common tasks, there are user role specific tasks. These include broadcasting a lecture if the user is an instructor and receiving a lecture if the user is a student. The user specific tasks are discussed after the common tasks section.

## 2. Start Writing

One of the very basic ways the instructor/student can use WriteOn1.0 is for annotating by enabling the scalable eVellum over an area of their choice. To enable the scalable eVellum, the user can follow the following steps:

i. Click the "Start Writing" button on the Main Form or click on the "Start Writing" icon on the toolbar. The "Start writing" button and icon are shown in Figure 28 and Figure 16, respectively.


**Figure 28: Start Writing Button**

ii. If the scalable eVellum is being enabled for the first time, the user has to choose a region over which they are most likely to annotate. The "Select Region" form gives the option of choosing to enable the eVellum over the full screen or just a particular area. A screenshot of this form is shown in Figure 13.

iii. If the user chooses the "Full Screen" option, then the eVellum deployed over the entire computer screen. If the user chooses the "Region" option, then the user selects the area of his/her choice and the eVellum is enabled only in that area. Once these choices are made, the WriteOn toolbar becomes visible if it is not already visible. The "Start Writing" icon on the toolbar is highlighted to green. The screenshot in Figures 29 and 17 shows the

scalable eVellum enabled over the entire desktop and over just a region, respectively. By default, the pen cursor appears as an orange box on the eVellum. Figure 17 shows a sample annotation. The WriteOn1.0 tool bar in Figures 29 and 17 have the "Start Writing" icon highlighted.

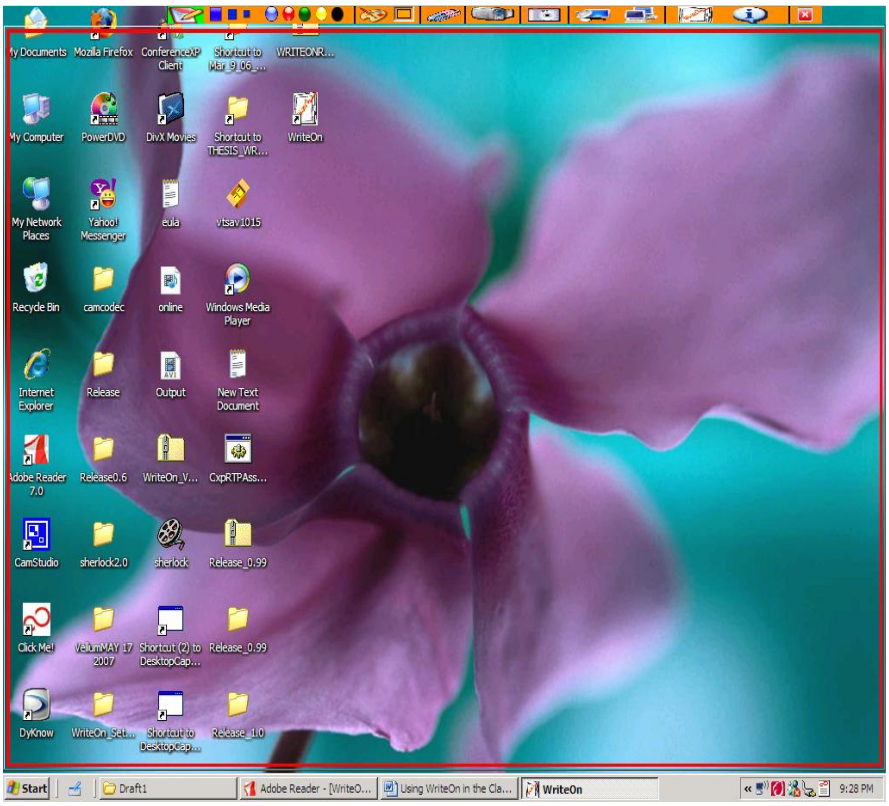



**Figure 29: Scalable eVellum enabled over entire desktop area**

### 3. Accessing the Desktop when Scalable eVellum is enabled

The eVellum can be toggled to access the user's Windows desktop or any application window under it. When the eVellum is toggled, its visibility is changed from visible to invisible if it is enabled, thus allowing the user to access the desktop or windows under the eVellum area. E-ink annotations are preserved and are re-displayed when the vellum is toggled back to its visible state.

Another operational mode of the eVellum is referred to as the *transparent mode*. In the transparent mode, WriteOn1.0 allows the user to access the window under the vellum using the electronic stylus while still displaying the e-ink in the forefront of the display. This mode is used for quick access to click on items in the underlying windows while still retaining the e-ink

display and ready access to continuing to apply e-ink. The steps to operate in transparent mode are:

i. Transparent Mode

The user can temporarily change the function of the vellum by right clicking on the area enclosed by the red border i.e., the area already selected as the location where e-ink will be applied. This changes the mouse cursor from that of a pen tip to the default cursor for the application running below the eVellum. Figure 30 illustrates the eVellum in pierce mode. As seen in this figure, the eVellum is still visible with all the annotation intact while the user performs actions on the window or desktop beneath the eVellum. We can resume writing on the vellum by moving the mouse cursor over the red border to enable the pen tip. This movement turns off the Transparent Mode.

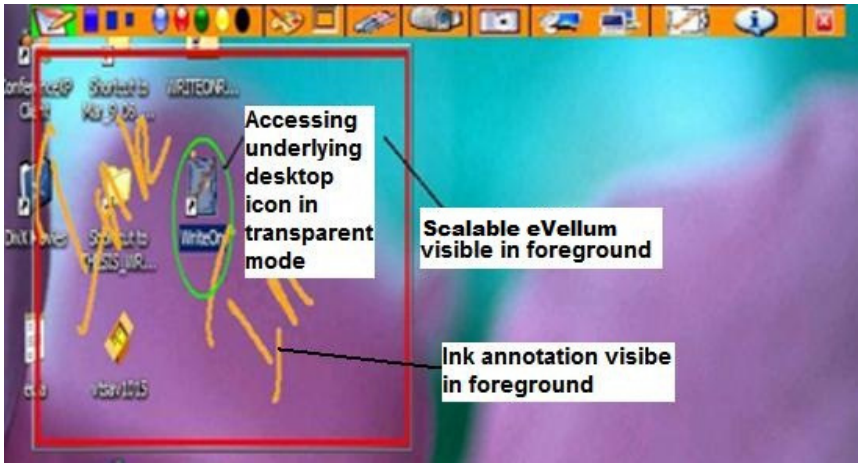



**Figure 30: Accessing desktop in the transparent mode of eVellum**

ii. Pierce mode

The vellum can be made invisible (if it is already active and visible) by clicking on the highlighted "Start writing" icon on the vellum tool bar. The vellum can alternately be inactive or invisible by using the Alt + F2 key combination in the PC mode of operation of the Tablet PC. In the Tablet mode, the user can program any one of the Tablet buttons to fire the Alt+F2 key combination when pressed by using the Tablet Buttons tab under the Tablet and Pen Settings window under Control Panel (See Section 18).

## 4. Setting Pen Color

The default pen color on the scalable eVellum is orange. The pen color can be changed to any one of the five basic colors (blue, red, green yellow and black) or a custom color from the WriteOn1.0 tool bar. Figure 31 shows the five basic color bubbles on the tool bar.


**Figure 31: Five basic color icons on WriteOn1.0 tool bar**

### 4.1 Choosing a color between blue, red, green, yellow and black

i. To choose from the colors blue, red, green yellow and black, click on the respective color bubble icon. The icon of the color chosen is highlighted with green. Figure 32 below shows the screenshot of the WriteOn toolbar and eVellum when the color is changed to red.

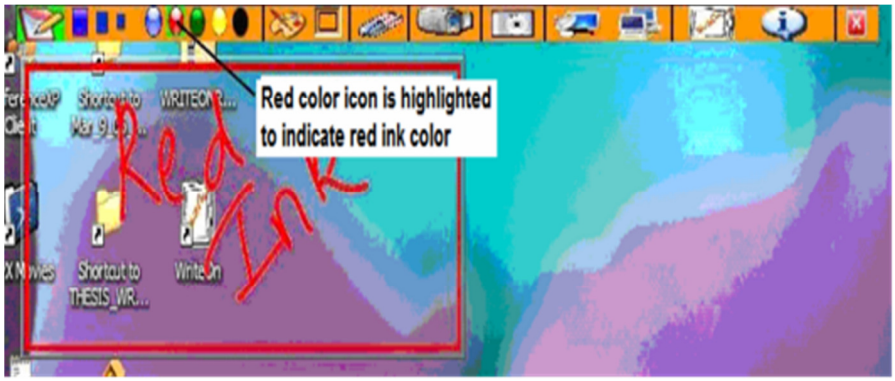



**Figure 32: Choosing Red Ink color**

ii. To revert to default color of Orange, click on the highlighted color bubble. The color bubble icon loses the highlight.

### 4.2 Choosing a custom color

i. WriteOn1.0 provides the option of choosing a custom color from the Windows Operating system colors palette. In order to choose a custom color, click on the paint palette icon as shown below in Figure 33.


**Figure 33: Custom color icon and Custom color box**

ii. This brings the custom color palette for the user to choose from as seen in Figure 34.

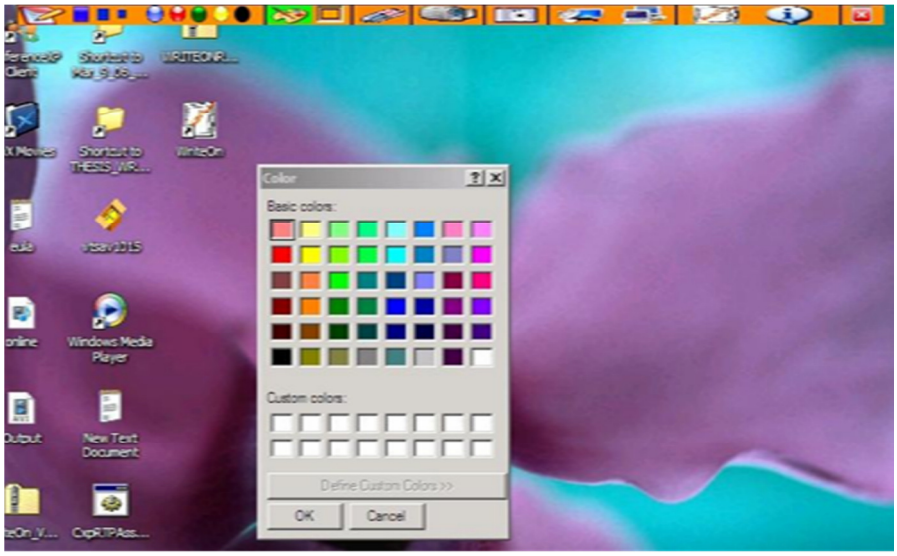



**Figure 34: Custom color palette**

iii. The user can choose the desired color by clicking on the color tile and then clicking OK from the bottom of the form. Figure 35 is the screenshot of the color palette when a custom color is chosen. As seen in Figure 35, the custom color tile has a dotted box around it when chosen.

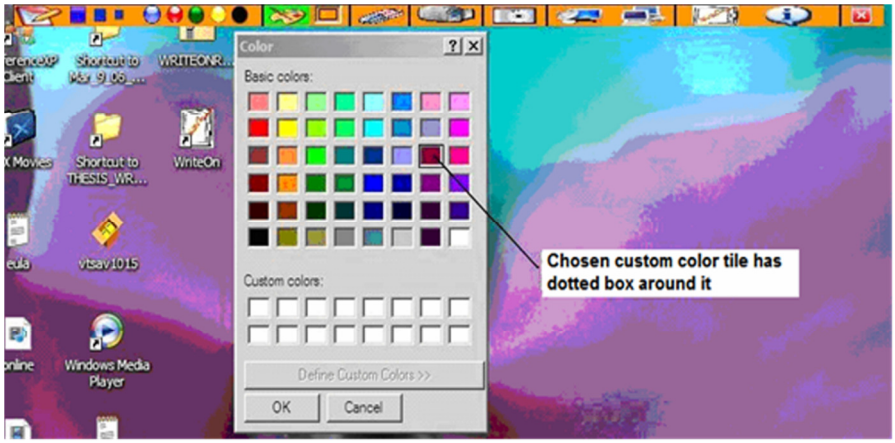



**Figure 35: Selecting a custom color**

iv. Upon clicking OK, the pen color reflects the custom color chosen. The custom color box shows the custom color chosen from the color palette. The custom color box is highlighted to green to indicate that the current pen color is that of the custom color. The WriteOn toolbar seen once a custom color is chosen is shown in Figure 36.

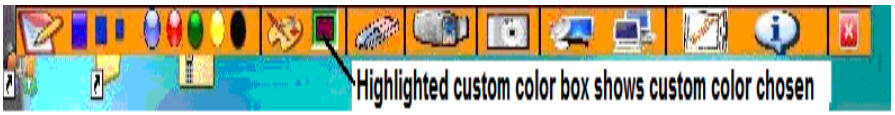



**Figure 36: Chosen custom color seen in the WriteOn1.0 toolbar**

## 5. Setting the Pen width

The instructor can choose to change the width of the e-ink according to his/her needs during a classroom presentation. WriteOn1.0 has four widths supported namely: default, narrow, medium and wide. The widths can be set by clicking on the appropriate width icon from the WriteOn toolbar. The icons used to set the pen widths are shown in Figure 37.


**Figure 37: Icons to set pen width on WriteOn1.0 toolbar**

As an example, if the instructor wants to set the pen width from default to wide and revert, he/she can follow the steps listed below.

i. Click the narrow width icon on the WriteOn1.0 tool bar. This changes the width of the pen and the size of the pen cursor.


**Figure 38: Selecting widest pen tip**

ii. The wide width icon appears as seen in the screenshot shown below in Figure 28. In order to revert to previous/original width, click on the wide width icon again. None of the width icons are selected as shown in Figure 39 when you revert to the default width of the pen tip.

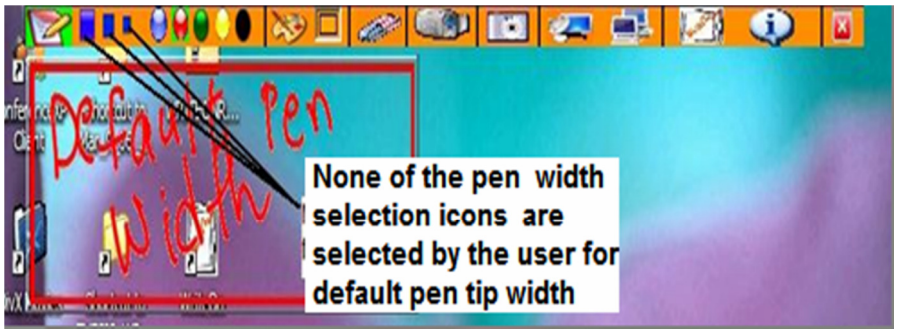



**Figure 39: Reverting to default pen tip width**

## 6. Snapshot Feature

The instructor/student can take a snapshot of any region on the user's Windows desktop with the *snapshot feature*. Snapshots are saved as JPEG files whose quality may be controlled by user settings. The instructor/student can use this feature to capture important data and notes as a picture file and place them into a specific folder, thus forming a class notebook.. The steps to take a snapshot are given below.

i. A snapshot of any region can be taken by clicking on the camera icon on the WriteOn1.0 toolbar as shown in the Figure 40.

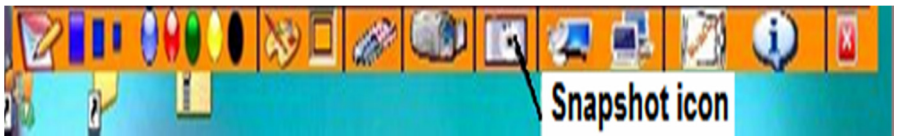



**Figure 40: Snapshot icon on WriteOn1.0 toolbar**

ii. The take snapshot form in Figure 41 appears and the user is asked to choose an area to capture and a quality setting for the image.

iii. Once the user chooses the area for the snapshot and its quality setting, the image is generated and saved by default in the "SnapShotCapture" folder in the home directory. The user can navigate and save the picture in any directory of his/her choice and can also rename the file as well. Figure 42 below shows the user saving a screenshot in the

"Lectures" folder. Subsequent captures are placed in the same folder by default and are given the same file name with a new sequence number appended. The user can select the "Remember settings for this session" option in case he/she wants to take several snapshots of the same area, for example, the area over which the scalable eVellum is deployed, in one session.

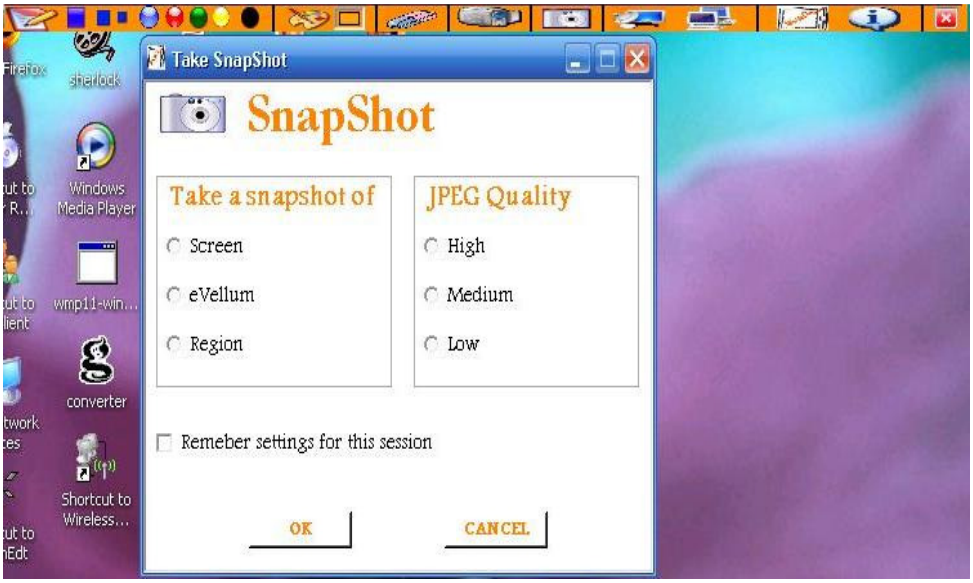



**Figure 41: Choosing area and quality for snapshot**

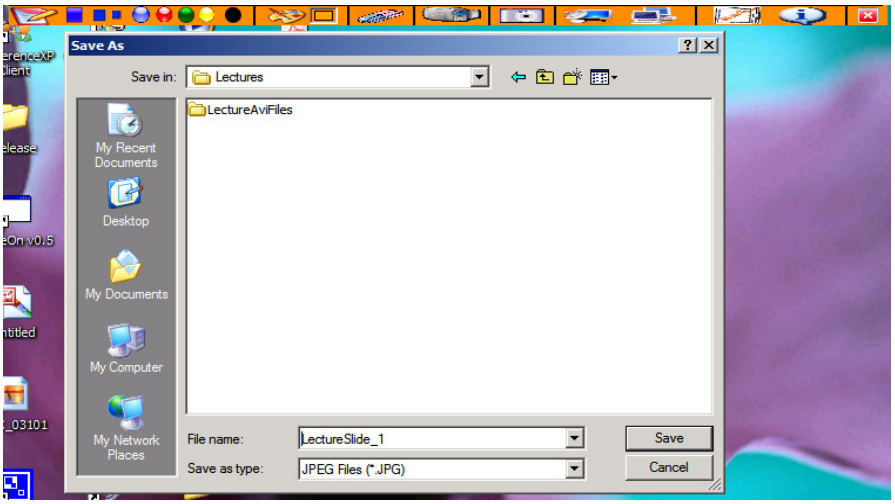



**Figure 42: Saving the snapshot**

## 7. Broadcast a Lecture

One of WriteOn1.0's important features when used as a teaching tool is the ability to broadcast, in real-time, a video of the desktop activities of the instructor's machine. The broadcast is done over a multicast sub-network to which both student and instructor are connected. Students can view the presentation material in a display window which renders the streamed video file. The following steps show how the instructor can start broadcasting a lecture.

i. The instructor can initiate the broadcast of a lecture by clicking the "Broadcast a Lecture" button from the Main Form or by clicking the "Broadcast a Lecture" icon in the tool bar. Figures 43 and 44 show the "Broadcast a Lecture" button and "Broadcast a Lecture" icon, respectively.


**Figure 43: Broadcast lecture button on Main Form**


**Figure 44: Broadcast a Lecture icon on WriteOn1.0 toolbar**

ii. In the next step, the instructor connects to any one of the two multicast venues. The instructor can choose a *friendly name* by which he/she would like to connect to the venue by editing the Name text box in the "Connect Venue" form. The default friendly name is the environment variable machine name which is set in the Windows My Computer Properties dialogue box. The "Connect Venue" form is seen in Figure 45 after the user has connected to Venue1 using the default friendly name. In the example below in Figure 45, the Tablet PC machine name is "LIFEBOOK".

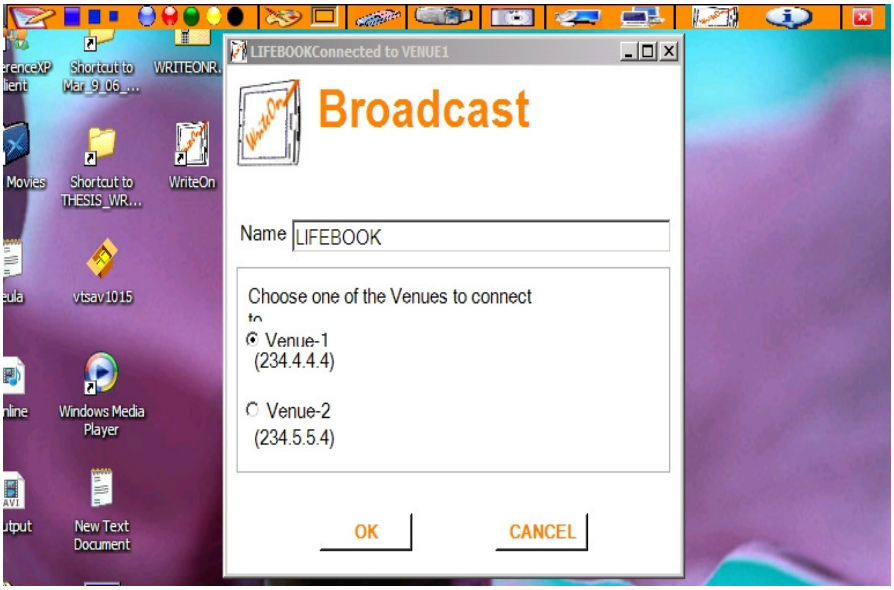



**Figure 45: Instructor choosing a venue for broadcast**

iii. Once the user is connected to a multicast venue/classroom, the next step is to select a region of interest to broadcast. The form to select a region appears if the user has not yet chosen an area of interest for enabling the vellum or screen recording as a movie. The select region form appears as shown in the Figure 46 below, which shows that the user has selected to broadcast only a region of the screen by selecting the "Region" option.

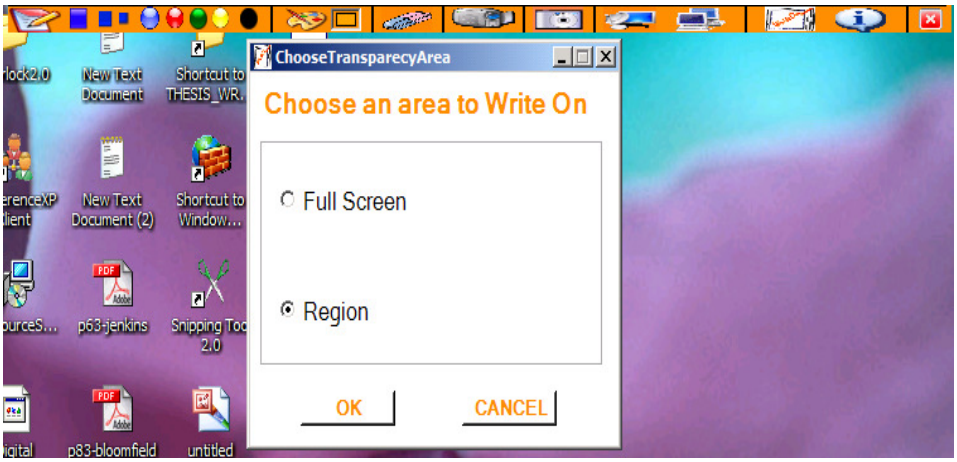



**Figure 46: Select region from for lecture broadcast**

iv. After the area of interest is selected, the desktop activities in the chosen area begin streaming to the students connected to the classroom/venue. The "Start Broadcast" icon on the toolbar is highlighted as green to indicate that a lecture is being broadcast as seen

in Figure 47. This figure also shows the scalable eVellum over the user selected area. Only the activities in the area encompassed by the scalable eVellum are broadcast.


**Figure 47: Lecture broadcast in progress**

## 8. Terminate a broadcast session

The instructor can terminate a broadcast session by clicking on the highlighted "Broadcast a Lecture icon". A dialog box appears confirming the termination of the broadcast and disconnection from the multicast venue. The disconnect dialog box looks like the one shown in Figure 48 below.

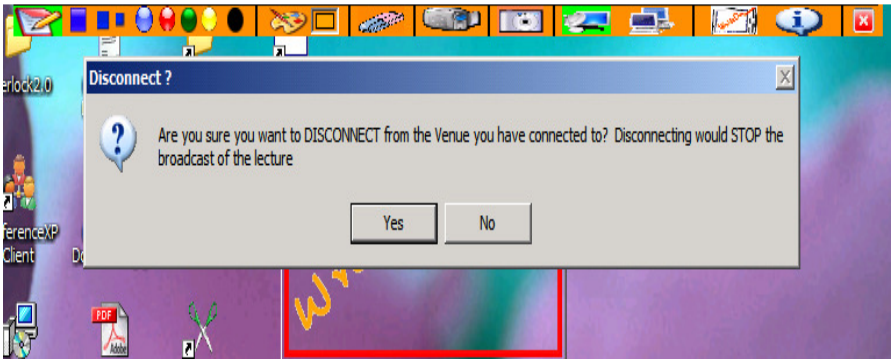



**Figure 48: Terminating a lecture broadcast session**

## 9. Save Lecture as Movie during Broadcast

The instructor can save the lecture he/she is presenting in any classroom session as a movie playlist. The movie playlist can then be shared online or reused during a future classroom session. The steps below illustrate how an instructor can broadcast, write and save a lecture at the simultaneously.

i. To begin saving material broadcast to the students, please click on the "Capture Screen activity" button in the Main Form or click the "capture screen activity" icon on the WriteOn tool bar. The button and icon are as seen in Figures 49 and 50.


**Figure 49: Start Screen recording button on Main Form**


**Figure 50: Start screen recording icon on WriteOn1.0 toolbar**

ii. The user then selects a region of interest to capture. The form to select a region appears if the user has not yet chosen an area of interest for enabling the vellum or for broadcasting. The *select region* form appears as shown in Figure 51.

iii. If the instructor has already chosen an area for the vellum or is currently broadcasting video, then the area chosen earlier becomes the default area from which the video file is generated. Once the region is chosen, the instructor is asked to choose a file name . The lectures are saved as movie playlists and the user can navigate to and save in any directory of his choice. The default directory is Lectures on the local home directory. The screenshot in Figure 52 shows the user saving the lecture with the name lecture1 on his/her desktop. The user can also choose to append the lecture to a previously saved playlist.


**Figure 51: Choosing area and quality of video for screen recording**

iv. After a name has been chosen, recording begins and the "start screen capture" on the tool bar is highlighted as seen in the screenshot in Figure 53 below. The screenshot also shows that the instructor is currently writing and broadcasting.

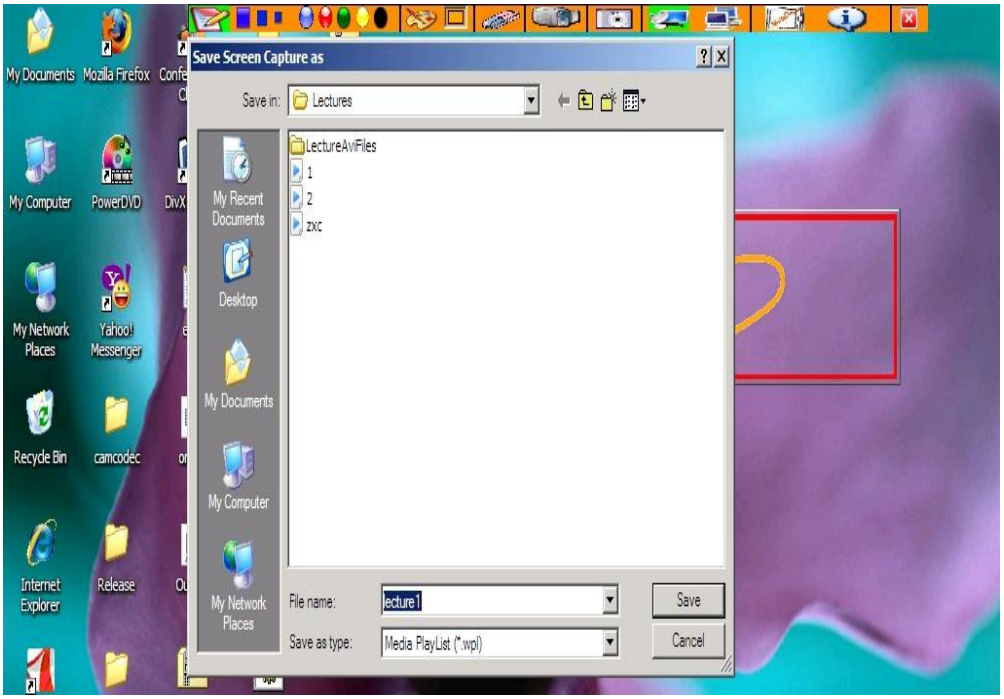



**Figure 52: Saving a screen recording session while broadcasting**

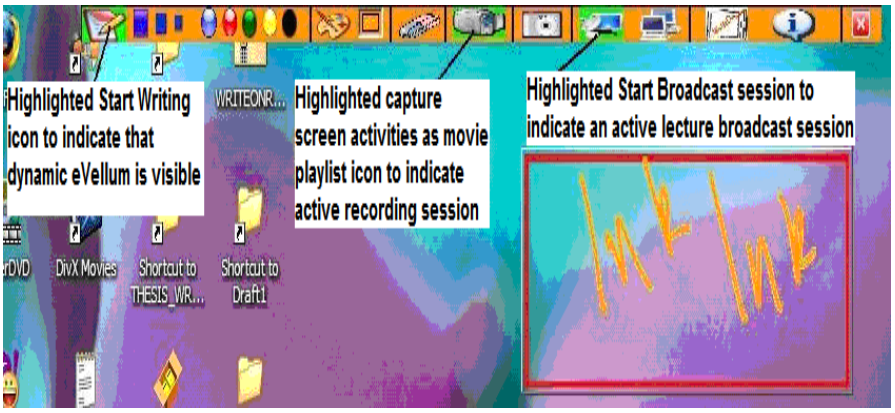



**Figure 53: Screen recording, broadcasting and annotation in progress**

## 10. Terminate Save Lecture Session during Broadcast

Stopping a *save lecture session* of the material being broadcast does not stop the broadcast process. Recording can be stopped by clicking on the highlighted "start screen capture" icon. This stops the current recording process and saves the lecture to the desired path. A message box pops up as seen in Figure 54 to indicate that the lecture is being saved.

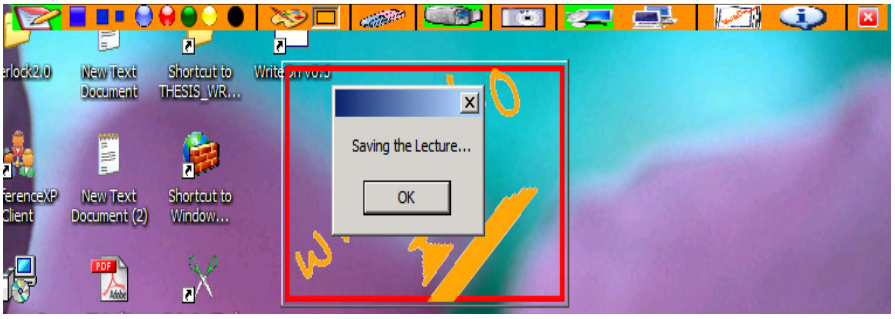



**Figure 54: Stopping a screen recording session**

## 11. Receiving a Lecture

A student can receive a live lecture being broadcast by connecting to the same multicast venue/classroom as the instructor. The live lecture stream is rendered in a separate window. The student has to connect to a multicast sub-network over which the instructor is broadcasting the lecture to receive it. There are two multicast venues/classrooms supported in WriteOn1.0. The instructor has to announce to the class which venue to connect to before a broadcast session

starts. Once the student knows the venue name he/she needs to connect to, he/she can start receiving the lecture by following the steps as follows.

i. The student must first connect to the appropriate venue/classroom from the "Receive Lecture" form by clicking on the "Receive Lecture" button on the Main Form or the icon on the toolbar. The "Receive Lecture" icon and the button are shown in Figure 55 and 56, respectively.


**Figure 55: Receive Lecture button on Main Form**


**Figure 56: Receive lecture icon on WriteOn1.0 toolbar**

ii. The student can change the friendly name by which he/she would like to connect to the classroom. The default friendly name is the machine name from the environment variable found in the Windows My Computer Properties dialogue box. The student must change the friendly name before clicking on the radio button corresponding to a venue. The screenshot of the receive lecture form in Figure 57, shows a student connected to "Venue-1" using the default name.

iii. Once the student is connected to the venue, they can click OK to start the receipt of the live lecture stream. The "Receive a Lecture" icon on the tool is highlighted to indicate the receipt of lecture. The live lecture stream is rendered in a separate window as seen in the screenshot shown below, Figure 58.

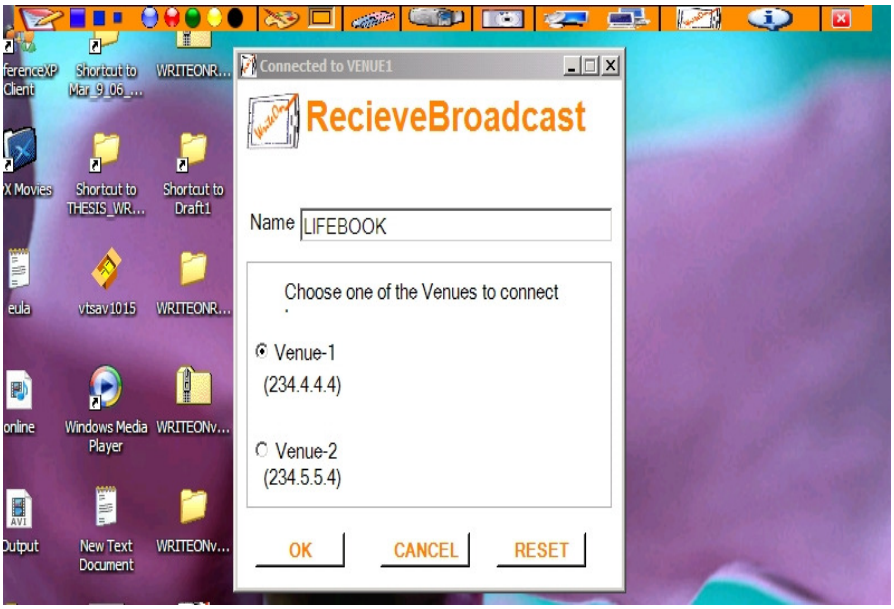



**Figure 57: Student choosing venue to Receive Lecture**

## 12. Terminate a receive Lecture Session

The student can terminate a receive lecture session by clicking on the highlighted “receive lecture icon”. A dialog box appears confirming the termination of the lecture receipt and disconnection from the multicast venue. The disconnect dialog box looks like the one shown below in Figure 59.

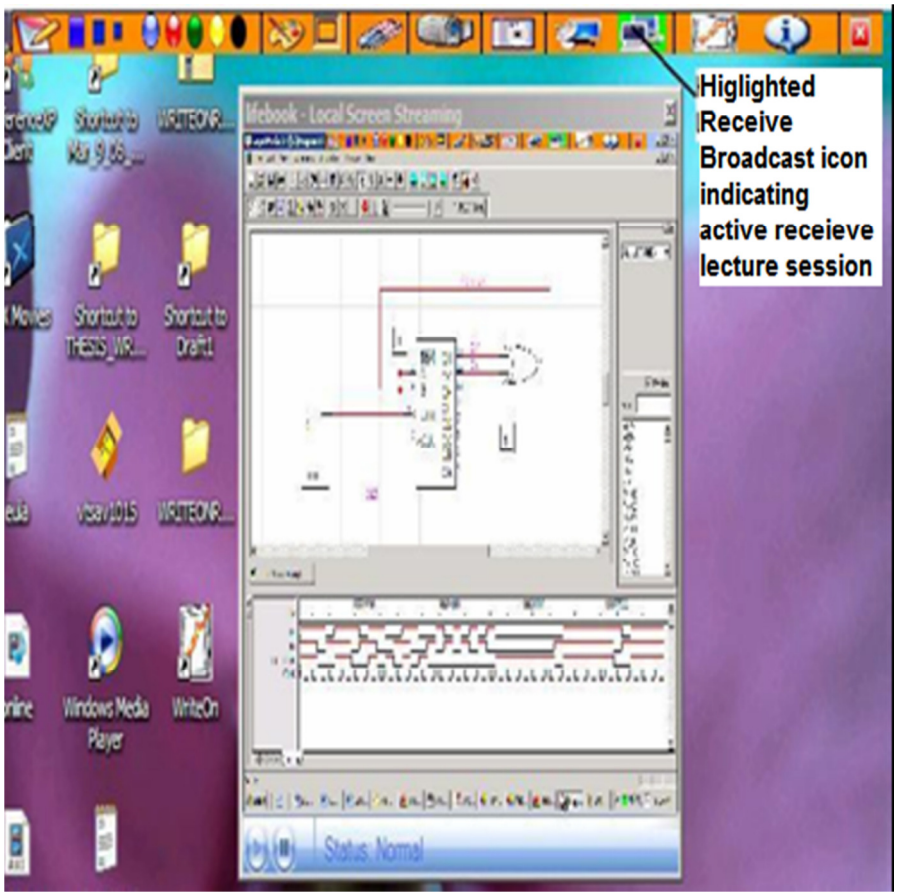



**Figure 58: Receive Lecture in progress**

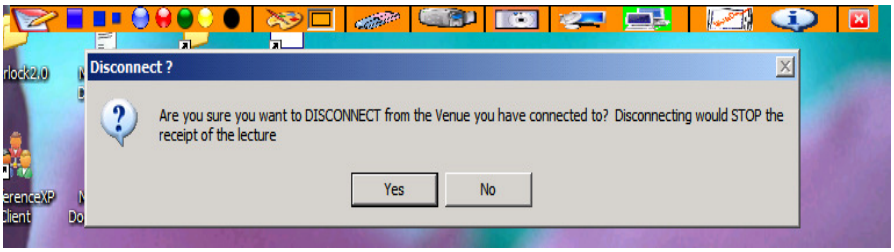



**Figure 59: Stop Receiving a Lecture**

### 13. Save Lecture as Movie and take Notes

A student may enable the dynamic vellum over the instructor's live lecture stream window, take notes with e-ink and save both the instructor's video and the student's notes. From the student user point of view, WriteOn1.0 can be used as an effective tool to make notes and for learning even when he/she is not receiving live lecture material. The students can enable the

dynamic vellum over any study material like PowerPoint presentations or simulation software and write over it. All the e-ink annotations along with video comprise the notes which can be captured as a movie playlist. The steps below illustrate how a student can receive, write on, and save a lecture.

i. To begin saving material broadcast by the instructor, click on the "Capture Screen Activity" button from the Main Form or the "capture screen activity" icon on the WriteOn tool bar. The button and icon are as seen in Figure 49 and Figure 50.

ii. Next select a region of interest to capture, e.g., the window in which the lecture is being rendered. The form to select a region appears as shown in the Figure 60 below if the student has not yet enabled the eVellum. .

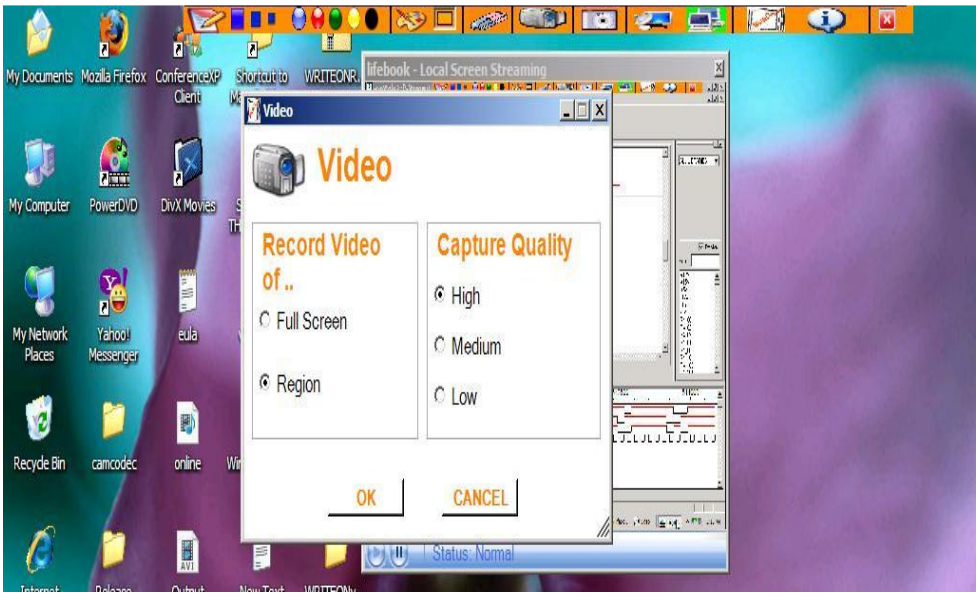



**Figure 60: Student choosing area and quality of recording lecture**

iii. If the student has already chosen an area over which the eVellum is enabled, then this area becomes the default area which will be recorded. Once the region is chosen, the student must choose a name by which they are going to save the lecture. Lectures are saved as movie playlists which can be navigated to and saved in any directory of their choice. The default directory is Lectures on the local home directory. The screenshot in Figure 61 shows the user

saving a lecture with the name "*lecture1*" in the Lectures folder. The user can also choose to append the lecture to a previously saved playlist.

iv. After a name has been chosen, recording begins and the "start screen capture" icon on the tool bar is highlighted as seen in the screenshot of Figure 62. Icons in the screenshot also indicate that the student is currently writing on the eVellum, receiving a lecture and recording the capture area.

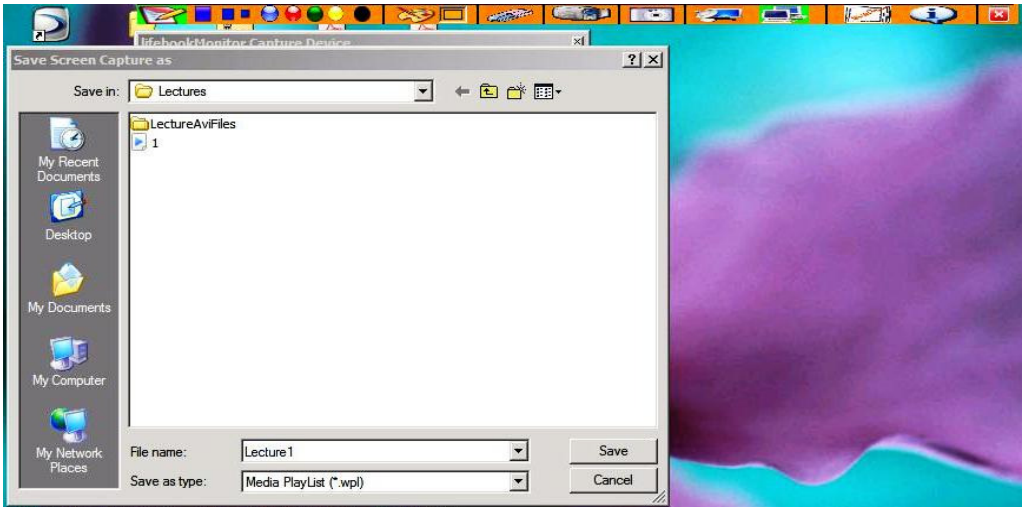



**Figure 61: Student choosing name of lecture file to record lecture while receiving it**

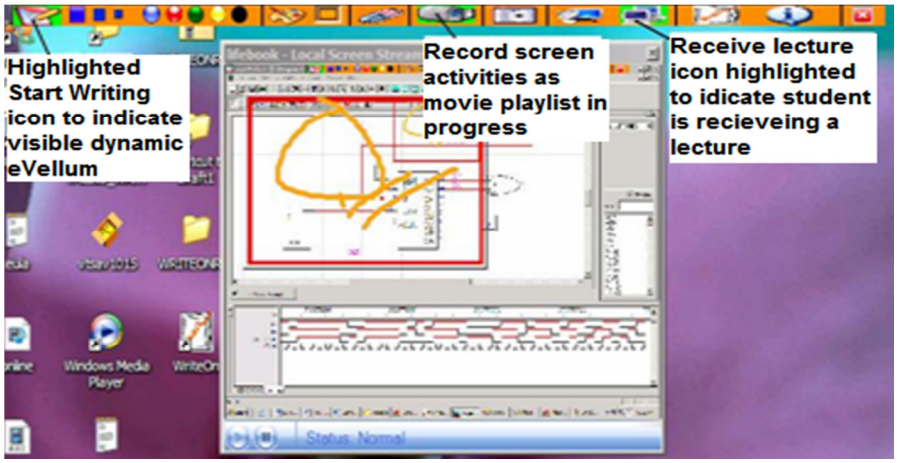



**Figure 62: Student receiving and recording lecture while writing on it**

**14. Stop Recording Lecture as a Movie during Receive Lecture Session**

Stopping the recording of a of the material being broadcast to the student does not stop the receiving process. That is, the broadcast lecture session will continue to be displayed on the student machine until the student clicks on the highlighted "Start screen capture" icon. Clicking this icon stops the current recording process and saves the lecture to the desired path. A message box will pop up as seen below in Figure 53 to indicate that the lecture is being saved.

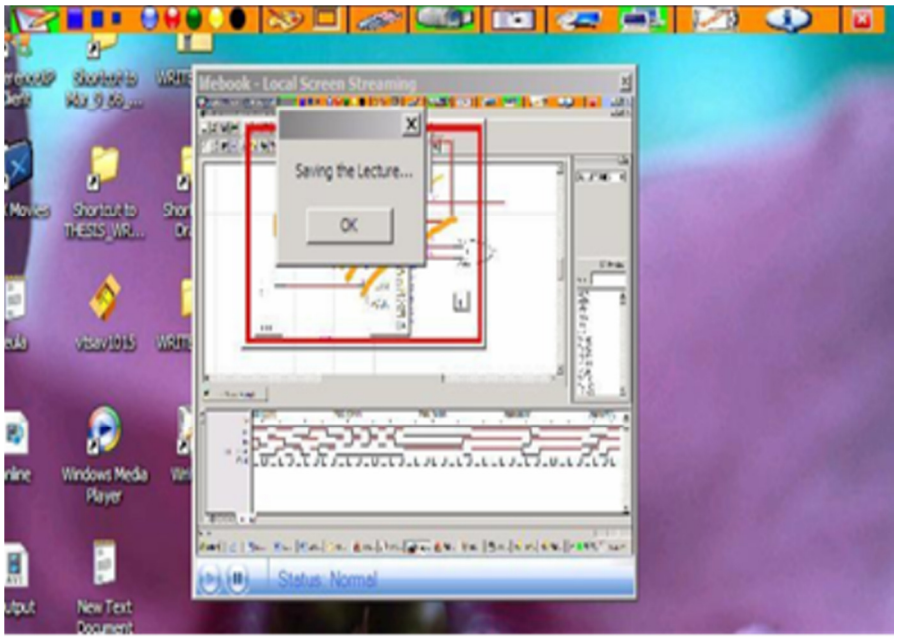



**Figure 63: Student stopping a record session of a received lecture stream**

**15. Erase Ink Annotations on Scalable eVellum**

The top of the Tablet PC pen stylus can be used to erase selected ink strokes from the scalable eVellum just like a normal eraser.(Unfortunately, not all manufacturer's pens are equipped with an eraser mechanism on the top of the stylus). In order to delete all the ink on the scalable eVellum, the user can use the Alt+F3 key combination or click the eraser icon on the toolbar as seen in Figure 64. The user can configure the Tablet PC buttons to perform the erase all ink function when pressed.


**Figure 64: Erase all ink icon on the WriteOn1.0 toolbar**

**16. Information about WriteOn1.0**

The user can view the details of the version of WriteOn tool and information on the developers of the application by clicking on the information icon on the toolbar as seen in Figure 65.


**Figure 65: WriteOn1.0 information icon on the WriteOn1.0 toolbar**

**17. Exit the WriteOn1.0 application**

To exit the WriteOn1.0 tool, the user can click on the close button located on the top right hand corner of the Main Form. When the toolbar is visible, the user has to click the exit icon as seen in Figure 66. Please note that you cannot exit the tool by clicking on the close button of the WriteOn1.0 Main Form when the toolbar is active.


**Figure 66: Exit WriteOn1.0 icon on the WriteOn1.0 toolbar**

**18. Set Tablet PC buttons for WriteOn1.0**

The user can set the Tablet PC buttons to map to certain action or a key or key combination when pressed. This becomes particularly useful to perform common actions like toggling the eVellum when using WriteOn1.0 in the tablet-mode. One of the Tablet buttons can be mapped to the Alt+F2 key combination that performs the toggle function. Also, since some Tablet-PC styluses do not come with an eraser attached to their back, it is essential that we provide an alternative mechanism to allow the user to erase ink on the scalable eVellum by mapping the tablet button to perform the Alt+F3 key combination. The steps to set the tablet buttons are as follows:

i. Access the Tablet and Pen Settings dialog from the Control Panel of Windows as seen in Figure 67.


**Figure 67: Selecting Tablet and Pen Settings icon from Control Panel**

ii. From the Tablet and Pen Settings dialog, navigate to the Tablet buttons tab. The screenshot of the Tablet buttons tab is shown in Figure 68.

iii. Since we want to map the tablet buttons to the Alt+F2 and Alt+F3 key combinations when operating in the tablet mode, we choose the Primary portrait option for the "Button settings for" menu item as seen in Figure Y. We then select a button to which we would like to map our key combination, for example "A Button" and then click the "Change" button.

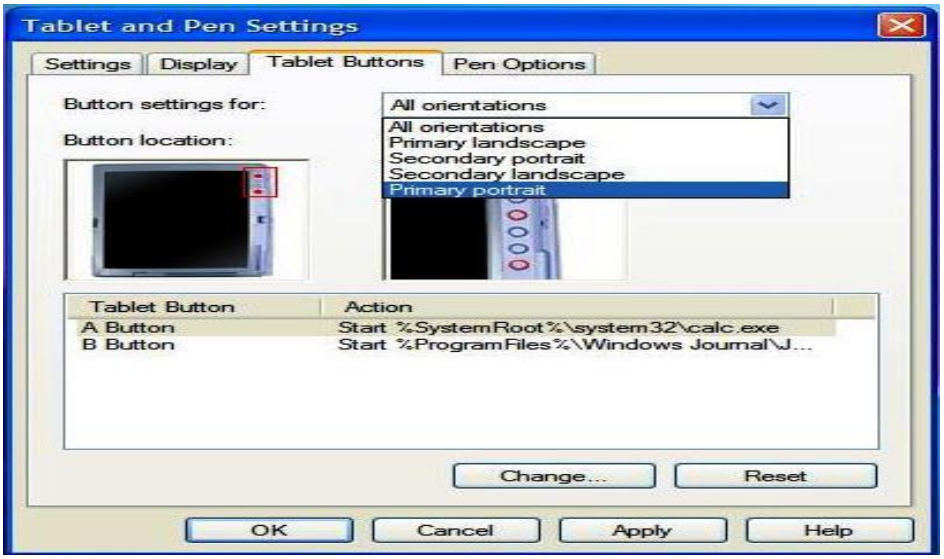



**Figure 68: Changing the "A Button" Action Settings**

iv. Upon clicking the “Change” button, the “Change Tablet Button Actions” dialog appears as seen in Figure 69.

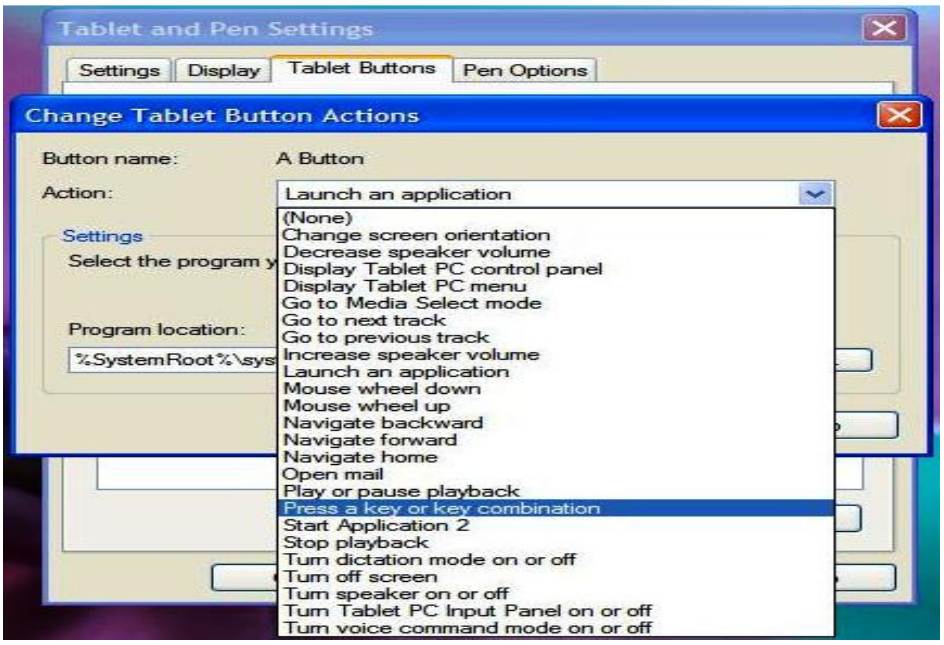



**Figure 69: Change Tablet Button Actions dialog**

v. Select the “Press a key or key combination” option against the “Action” menu item. The dialog now appears as shown in Figure 70.

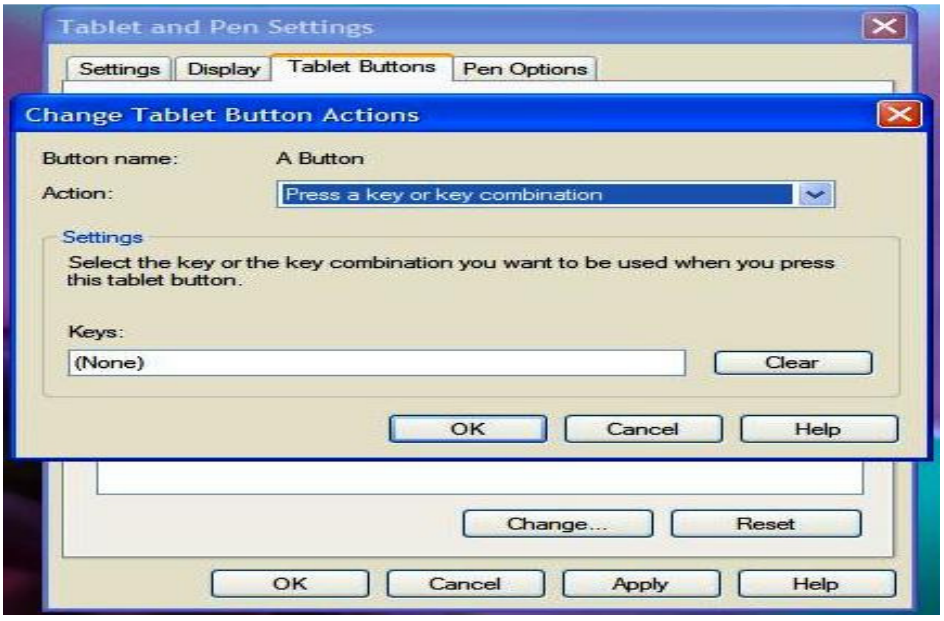



**Figure 70: Selecting the “Press a key or key combination” from Change Tablet Button Actions dialog**

vi. Then go to the textbox below the Keys menu item, press the key combination you desire to map the button to. In the case of WriteOn1.0, we select the A button to map to the Alt+F2 combination by pressing down on the Alt+F2 keys until the combination appears in the textbox as shown in Figure 71.

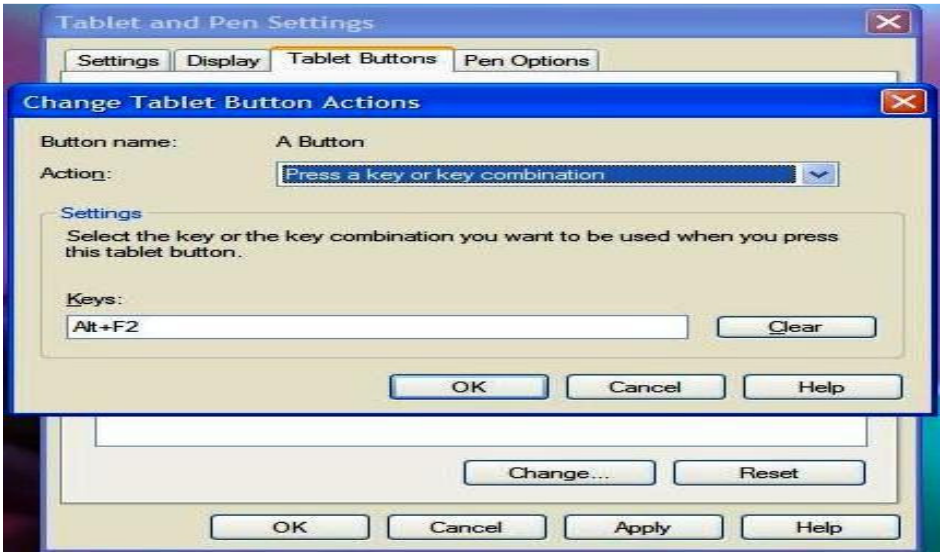



**Figure 71: Changing A Button's action to Alt+F2**

vii. In order to map the Alt+F3 combination to the other tablet button, the B button, we go back to the Tablet Buttons tab and select the B button and then follow steps iv, v and vi to map the B button to the Alt+F3 key combination. Figure 72 shows the B button mapped to the Alt+F3 key combination.

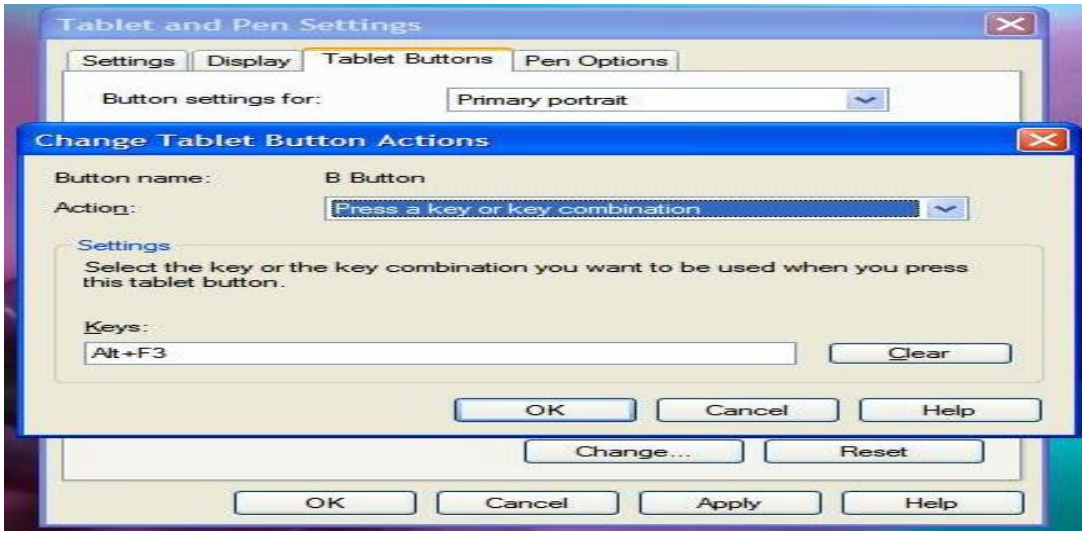



**Figure 72: B Button mapped to Alt +F3 key combination**

**19. Uninstall WriteOn1.0**

The user has to uninstall any previous versions of WriteOn installed on his/her computer to proceed with the installation of WriteOn1.0. The WriteOn1.0 setup wizard detects for any previous installations of WriteOn and gives the user the option of uninstalling it.

The user can select the "Remove WriteOn" option from the WriteOn setup wizard as seen in Figure 73 and clicking on the "Finish" button of the dialog. Figure 74 shows the progress in uninstalling the tool. Once the tool is removed, the setup wizard dialog displays the "WriteOn has been successfully removed" message as seen in figure 75. The user has to select the "Close" button from the dialog.

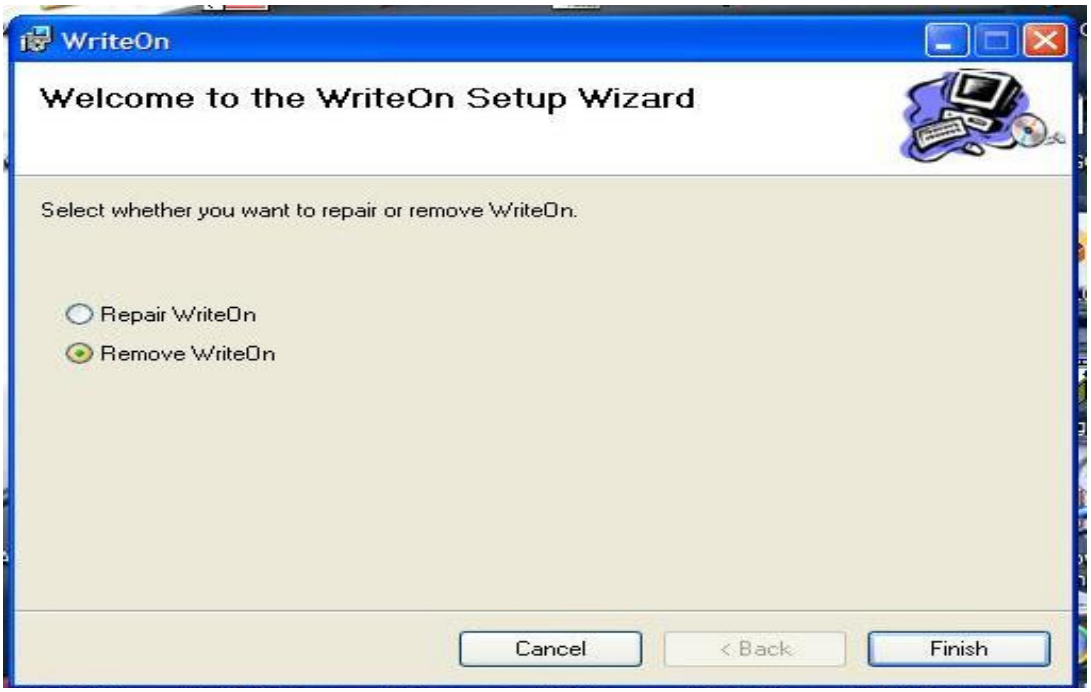



**Figure 73: Selecting Remove WriteOn option from WriteOn Setup Wizard**

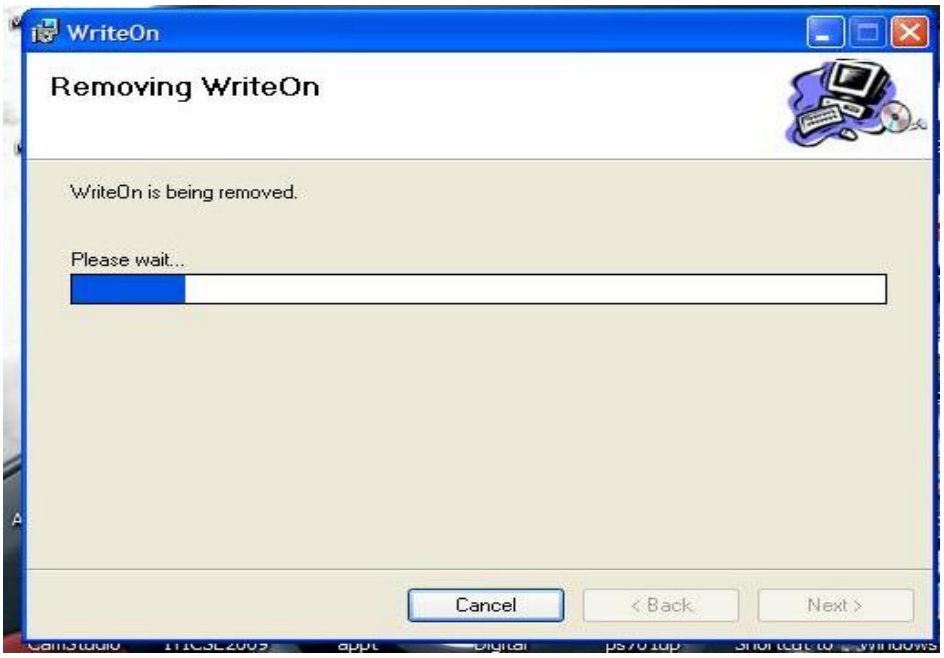



**Figure 74: Progress of uninstalling WriteOn**

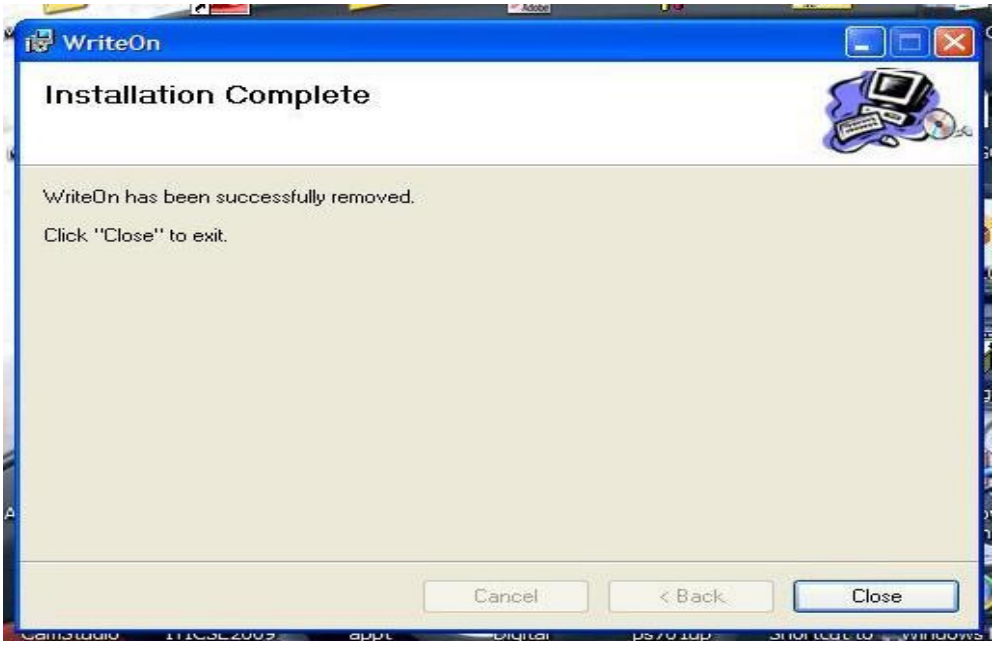



**Figure 75: Completion of WriteOn uninstallation**

# Appendix II

## Moving and resizing a transparent borderless form

The code in this appendix describes the source code that enables us to move and resize a transparent borderless form, creating the scalable eVellum. This code can be found in the file named "OverlayVellum.cs" in the "Transparency" folder of the source code. The WndProc function implements the logic behind creating a resizable scalable eVellum. The WndProc is a Windows defined callback function that receives all the input messages directed at that particular form. Whenever any event occurs on the form like a resize, click, mouse move etc., Windows will call this function. The Message parameter contains the message sent to the target window/form. The Message contains the details of the mouse events like its position, mouse button clicked on etc. that can used to detect events on the window like a mouse down and mouse move. The WndProc function implements the logic that creates the scalable eVellum. The comments in the code snippet explain the implementation of the scalable eVellum.

```
protected override void WndProc(ref Message m)
{

  // WE START LISTENING TO THE MOUSE EVENTS IN ORDER TO
  // DETECT A POTENTIAL RESIZE EVENT OR MOVE EVENT.
  // EVENTHOUGH THE OVERLAY VELLUM FORM HAS A RESIZABLE
  // BORDER, WE CANNOT DETECT THE START AND STOP OF A
  // RESIZE EVENT.WE CAN ONLY DETECT A CHANGE IN SIZE AND
  // LOCATION OF THE FORM. THIS IS BECAUSE A WINDOW CAN
  // ONLY DETECT EVENTS/ACTIONS THAT TAKE PLACE IN ITS
  // CLIENT AREA WHICH DO NOT INCLUDE THE BORDERS.

  // WE WANT TO HANDLE MESSAGES THAT ARE TARGETED
  // FOR THE OVERLAY VELLUM OF THE DYNAMIC EVELLUM.
  // WE ACCOMPLISH THIS BY COMPARING THE VALUE OF THE HWnd
  // PROPERTY OF THE Message STRUCTURE THAT STORES THE
```

```
// VALUE OF THE HANDLE OF THE WINDOW TO THE HANDLE VALUE
// OF THE OVERLAY VELLUM.
if ( m.HWnd == this.Handle )
{
  // WE BEGIN BY FETCHING THE COORDINATES OF THE MOUSE
  // POSITION. THE INTEGER OBJECTS STORING THE MOUSE
  // COORDINATES ARE STORED IN THE LParam PROPERTY OF THE
  // Message STRUCTURE.

  // WE FIRST CONVERT THE INTEGER TO A 32-BIT
  // SIGNED INTEGER AND THEN USE THE GET_X_LPARAM
  // FUNCTION TO EXTRACT THE X COORINATE AND
  // GET_Y_LPARAM FUNCTION TO EXTRACT THE Y COORDINATE.
  // THESE FUNCTIONS ARE DESCRIBED AFTER THE WNDPROC
  // FUNCTION.

  int xPos = GET_X_LPARAM(m.LParam.ToInt32());
  int yPos = GET_Y_LPARAM(m.LParam.ToInt32());

  // THE NEXT STEP IS TO DETECT IF THE MOUSE CURSOR IS AT
  // THE RED BORDER OF THE OVERLAY EVELLUM
  // IF THE MOUSE IS AT THE BORDER, IT INDICATES A
  // POTENTIAL RESIZE EVENT. WE DETECT THIS BY FIRST
  // CALCULATING THE DISTANCE OF THE MOUSE
  // COORDINATES FROM THE TOP AND LEFT EDGES OF THE
  // OVERLAY EVELLUM.

  double xIn = xPos - this.Left;
  double yIn = yPos - this.Top;
```

```
// WE THEN CALCULATE THE DISTANCE OF THE MOUSE
// COORDINATES FROM THE BOTTOM AND RIGHT EDGES OF THE
// OVERLAY EVELLUM

double xR = this.Right - xPos;
double yR= this.Bottom - yPos;

// THE FIRST STEP IN THE SEQUENCE OF STEPS TO PERFORM A
// RESIZE ACTION IS THE MOUSE DOWN EVENT AT THE BORDER
// OF THE FORM. WE CAPTURE THE MOUSE DOWN EVENT ON THE
// OVERLAY EVELLUM RED BORDER BY CHECKING THE Msg
// PROPERTY OF THE Message STRUCTURE. IF THE VALUE OF
// THE Msg PROPERTY CORRENSPONDS TO a1 OR A
// WM_NCMOUSEDOWN MESSAGE IT MEANS A MOUSE DOWN
// EVENT HAS OCCURRED

if ( m.Msg == 0xa1 ) // A MOUSE DOWN EVENT HAS OCCURRED
{

  // IF THE CHILD OVERLAY EVELLUM IS NOT VISIBLE AS
  // WHEN OPERATING IN THE TRANSPARENT MODE, THEN MAKE
  // IT VISIBLE

  if ( this.childOverlay.Visible == false )
  {
     this.childOverlay.Visible=true;
  }
```

```
// WE COMPUTE THE POSITION OF MOUSE IN TERMS OF
// SCREEN COORDINATES. THIS IS BECAUSE THE MOUSE
// COORDINATES STORED IN THE Message STRUCTURE USE
// THE TOP LEFT CORNER OF THE OVERLAY EVELLUM AS THE
// REFERENCE POINT AND HENCE THEY NEED TO BE
// CONVERTED TO ABSOLUTE LOCATION ON THE SCREEN

Point pt = this.PointToScreen (new Point
                                (m.LParam.ToInt32()));

// WE CREATE AN INSTANCE OF THE MOUSEEVENT ARGUMENTS
// CLASS WITH THE SCREEN COORDINATES OF THE MOUSE
 // CLICK WE SHALL PASS THE ARGUMENTS OBJECT AS A
// PARAMETER TO THE OnMouseDown EVENT IN THE OVERLAY
// EVELLUM WHERE WE SET THE VARIABLE
// IsVellumResizeMode TO TRUE TO INDICATE THE START
// OF A RESIZE EVENT.
// THE OnMouseDown EVENT HANDLER IS ALSO PRESENTED
// IN THIS APPENDIX

MouseEventArgs ev =
                new MouseEventArgs(MouseButtons.Left,
                                    0,pt.X, pt.Y, 0);

// CHECK IF MOUSE IS AT ANY OF THE BORDERS BY
// CHECKING IF THE MOUSE IS AT PIXEL DISTANCE
// OF LESSER THAN 5 PIXELS FROM ANY ONE OF THE
```

```
  // EDGES OF THE OVERLAY EVELLUM.

  if ((xIn > 5 && yIn < 5) ||
     (yIn > 5 && xIn < 5) ||
      xR  < 5 || yR < 5 )
  {
    // IF WE ARE AT THE BORDER TRIGGER THE OnMouseDown
    // event TO INDICATE START OF A POTENTIAL RESIZE
    // EVENT
    this.OnMouseDown(ev);
  }
}
// THE NEXT SEQUENCE FOR A EVELLUM RESIZE AFTER A CLICK
// AT THE BORDER IS THE MOUSE MOVE EVENT. WE CHECK IF A
// MOUSE MOVE EVENT OCCURS AFTER THE MOUSE IS CLICKED
// AT THE BORDER. THE VARIABLE IsVellumResizeMode THAT
// IS SET IN THE OnMouseDown EVENT HANDLER INDICATES IF
// WE ARE THE EVELLUM IS IN RESIZE MODE.
// THE MOUSE MOVE EVENT IS DETERMINED BY COMPARING THE
// Msg PROPERTY OF THE Message STRUCTURE WITH A VALUE
// OF 0xa0 OR WM_NCMOUSEMOVE MESSAGE

else if (m.Msg == 0xa0) // WM_NCMOUSEMOVE
{
  // WE CHECK IF THE IsVellumResizeMode VARIABLE IS SET
  // DETERMINE IF WE ARE ACTUALLY RESIZING.

  if (this.IsVellumResizeMode )
```

```
{
  // WE RESET THE VALUE TO FALSE BECAUSE WE HAVE
  // ALREADY STARTED RESIZING.
  this.IsVellumResizeMode = false;

  // WE COMPUTE THE SCREEN COORDINATES OF THE MOUSE
  // POSITION AND CREATE MouseEventArgs INSTANCE
  Point pt =
    PointToScreen(new Point(m.LParam.ToInt32()));
  MouseEventArgs ev =
    new MouseEventArgs( MouseButtons.Left, 0,
                        pt.X,pt.Y, 0);

  // WE NEED TO ENSURE THAT WE ARE AT THE BORDER
  // WHILE PERFORMING THE RESIZE. IF WE DEVIATE FROM
  // THE BORDER, WE SHALL GO INTO THE WRITING MODE
  // AND THE POTENTIAL RESIZE EVENT NO LONGER REMAINS
  // VALID. SINCE THE OVERLAY VELLUM HAS A RESIZABLE
  // BORDER IN THE DESIGN OF THE DYNAMIC EVELLUM, A
  // MOUSE MOVEMENT AUTOMITICALLY RESIZES IT.
  // THE CHILD OVERLAY'S SIZE ALSO CHANGES TO THE
  // SAME DIMENSIONS AS THAT OF THE PARENT OVERLAY.
  if((xIn < 5 || yIn < 5 || xR < 5 || yR < 5))
  {
    // THE ONMOUSEUP EVENT HANDLER IS NECESSARY TO
    // NOTIFY THE COMPLETION OF THE REDIMENSIONING OF
    // THE DYNAMIC EVELLUM BECAUSE WINDOWS DOES NOT
    // HAVE EVENT HANDLERS TO DETECT IT. SINCE THE
    // DIMENSIONS OF THE SCREEN CAPTURE AREA IS THE
```

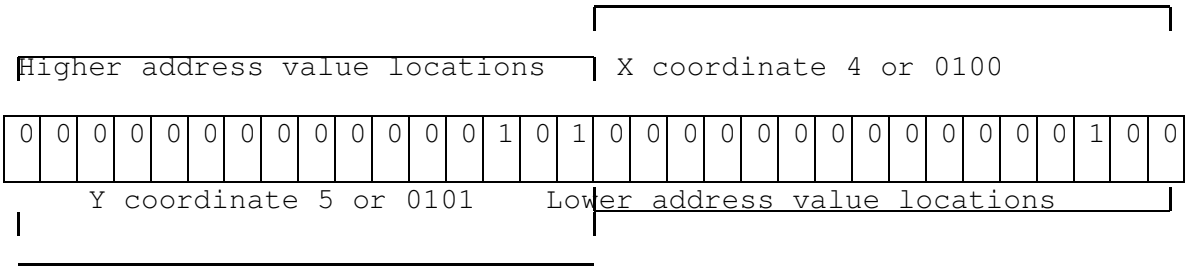
```
        // SAME AS THAT OF THE DYNAMIC EVELLUM,
        // WRITEON1.0 HAS TO MODIFY THE SCREEN
        // CAPTURE DIMENSIONS ONCE THE DYNAMIC EVELLUM IS
        // RESIZED.
        this.OnMouseUp(ev);
      }
    }
  }
base.WndProc(ref m);

// IN ORDER TO MOVE THE DYNAMIC WHEN THE USER CLICKS ON
// THE RED BORDER WE HAVE TO TRICK THE OVERLAY EVELLUM
// WINDOW TO BELIEVING THAT THE USER HAS CLICKED
// ON THE WINDOW TITLE BAR. ONCE A WINDOW HAS DETECTED
// THAT A CLICK HAS OCCURRED ON THE TITLE BAR, IT
// TRIGGERS A MOVE WINDOW EVENT WHEN THE MOUSE IS MOVED,
// CREATING THE DRAG DROP EFFECT ON THE DYNAMIC EVELLUM
// WE FIRST CHECK TO SEE IF THE USER HAS CLICKED ON THE
// RED BORDER BY TESTING FOR THE m.Msg VALUE. A VALUE OF
// 0X84 INDICATES THAT THE USER HAS CLICKED ON THE
// OVERLAY EVELLUM.

if (m.Msg == 0x84)
{
  // ONCE WE HAVE DETECTED A CLICK OF THE MOUSE, WE
  // DETECT OF IT WAS ON THE CLIENT AREA OF THE FORM
  // COMPRISING THE RED BORDER BY TESTING FOR THE m.Msg
  // VALUE. A VALUE OF 0x1 INDICATES THAT THE USER HAS
```

```
// CLICKED ON THE CLIENT AREA OVERLAY EVELLUM.

if (m.Result.ToInt32() == 0x1) // HTCLIENT
{
  // UPON CLICKING ON THE BORDER, WE DETECT A POTENTIAL
  // DRAG AND DROP EVENT WHICH OCCURS WHEN THE USER
  // CLICKS ON THE TITLE BAR OF ANY WINDOW.

  this.isMoveMode = true;

  // SINCE THE RED BORDER IS 5 PIXELS WIDE, WE HAVE TO
  // ENSURE THAT THE MOUSE CURSOR IS AT A DISTANCE OF
  // LESS THAN 10 PIXELS FROM THE EVELLUM EDGES SINCE
  // THE RESIZABLE FORM BORDER IS OF 5 PIXELS.

  if (xIn<10 ||yIn<10 || xR<10 || yR<10)
  {
    // REENABLE CHILD VELLUM WHEN MOUSE IS HOVERED ON
    // THE BORDER TO REACTIVATE THE EVELLUM FOR WRITING

    if(this.childOverlay.Visible==false)
    {
      this.childOverlay.Visible=true;
    }

    // ELSE WER ARE ON THE BORDER, TRIGGER A DYNAMIC
    // EVELLUM MOVE THE EVELLUM BY SENDING WM_HTCAPTION
    // MESSAGE TO THE WINDOW
```

```
        m.Result = (IntPtr)0x2;
      }
    }
    else
    {
      this.isMoveMode = false;
    }
  }
}

// THE GET_X_LPARAM FUNCTION RETURNS THE X COORDINATE OF
// MOUSE POSITION. IT IS STORED IN THE LOWER ORDER INTEGER
// OF THE LPARAM VALUE. INORDER TO RETRIVE IT, WE PERFORM A // BITWISE
AND OPERATION OF THE LPARAM VALUE WITH 0Xffff.

public static Int32 GET_X_LPARAM(int lParam)

{
     return (lParam & 0xffff);

}

// THE GET_Y_LPARAM FUNCTION RETURNS THE Y COORDINATE OF  // MOUSE
POSITION. IT IS STORED IN THE HIGHER ORDER INTEGER
// OF THE LPARAM VALUE. INORDER TO RETRIVE IT, WE PERFORM A // BITWISE
OPERATION OF SHIFTING THE BITS IN THE LPARAM
// VALUE BY 16.
```

```
public static Int32 GET_Y_LPARAM(int lParam)
{
      return (lParam >> 16);

}
```

The diagram below illustrates how the X and Y coordinates are extracted from the LParam value of the Message structure. Suppose that the coordinates of the mouse position are X = 4 and Y= 5. When represented as a 32-bit integer in which the X coordinate occupies the lower 16 bits (lower address values) and the Y coordinate is stored in the upper 16 bits of the 32 bit integer (higher address values), the LParam variable storing the X and Y coordinates would look like the diagram below




The GET_X_LPARAM function performs a bit wise AND operation of the above 32-bit integer with 0xffff. The result of the bitwise operation is 4 or the X- coordinate as shown below.

```
      0000 0000 0000 0101 0000 0000 0000 0100
      0000 0000 0000 0000 1111 1111 1111 1111
      ____________________________________________

      0000 0000 0000 0000 0000 0000 0000 0100 = 4
      ____________________________________________
```

The GET_Y_LPARAM function performs a bit wise shifting of the 32-bit integer, the LParam variable by 16 to the right. The result of the bitwise operation is 5 or the Y- coordinate as shown below:

```
0000 0000 0000 0101 0000 0000 0000 0100 >> 1 is
0000 0000 0000 0010 1000 0000 0000 0010
```

Performing the bit shift operation 16 times gives us the Y coordinate as seen below:

```
0000 0000 0000 0000 0000 0000 0000 0101 = 5
```

```
protected override void OnMouseDown(MouseEventArgs e)
{
 // WE STORE THE DIMENSIONS OF THE PREVIOUS SIZE OF THE
 // EVELLUM BEFORE THE RESIZE ACTION BECAUSE WE WANT TO
 // PERFORM A CHANGE IN THE DIMENSIONS OF ANY ONGOING
 // SCREEN CAPTURE ACTIVITY ONLY IF THE DIMENSIONS HAVE
 // CHANGED BY ATLEAST 10 PIXELS ON EITHER THE WIDTH OR THE
 // HEIGHT.
  this.prevClientRect = this.ClientRectangle;

 // WE SET THE MODE TO A VELLUM RESIZE MODE TO INDICATE TO
 // THE WINDOW THAT WE ARE PERFORMING A RESIZE.THE WND PROC
 // FUNCTION CHECKS FOR THIS VARIABLE WHEN A MOUSE MOVE
 // EVENT OCCURS TO PERFORM A RESIZE OPERATION.

     this.IsVellumResizeMode = true;

 // WE SET THE FONT OF THE OVERLAY VELLUM TO GARAMOND. WHEN
```

```
 // A CHANGE IN FONT EVENT IS DETECTED BY THE
 // CAPTURE CONTROLLER OF WRITEON, IT CHECKS IF THE FONT IS
 // ARIAL IN ORDER TO PERFORM UPDATES TO THE
 // CAPTURE DIMENSIONS FOR AN ONGOING SCREEN RECORDING
 // OR BROADCAST EVENT. OTHERWISE, NO ACTION IS PERFORMED
 // UPON A CHANGE IN THE FONT.

  this.Font = new System.Drawing.Font("Garamond",10.0F,
                          System.Drawing.FontStyle.Bold);
  base.OnMouseDown (e);
}

protected override void OnMouseUp(MouseEventArgs e)
{

 // WE SET THE PAINTED VARIABLE TO FALSE TO ENSURE THAT THE
 // STROKES ARE RENDERED ON THE DYNAMIC EVELLUM WHEN THE
 // OVERLAY EVELLUM REPAINTS ITSELF WHEN IT'S SIZE CHANGES.

    this.painted =false;

 // WE COMPARE THE CHANGE IN SIZE OF THE DYNAMIC EVELLUM
 // TO ENSURE THAT EITHER THE HEIGHT OR WIDTH HAS CHANGED
 // BY ATLEAST 10 PIXELS BECAUSE WE WANT TO CHANGE THE
 // DIMENSIONS OF THE CAPTURE AREA ONLY IF IT IS
 // SIZABLE.
```

```
  if (( Math.Abs (this.prevClientRect.Height - this.Height)
           >= 10 || ( Math.Abs(this.prevClientRect.Width -
           this.Width) >=10)))
  {
    // WE SET THE FONT OF THE CHILD OVERLAY TO ARIAL. WHEN
    // A CHANGE IN FONT TO ARIAL EVENT IS DETECTED BY THE
    // CAPTURE CONTROLLER OF WRITEON, IT UPDATES THE
    // CAPTURE DIMENSIONS FOR AN ONGOING SCREEN RECORDING
    // OR BROADCAST EVENT

     this.Font = new System.Drawing.Font ("Arial",10.0F,
                       System.Drawing.FontStyle.Bold);
  }

 // WE HAVE TO RESET THE MOVE MODE TO FALSE TO INDICATE
 // THAT WE ARE NOT PERFORMING A DRAG AND DROP ACTION ON
 // THE EVELLUM

  this.isMoveMode=false;

 // WE HAVE TO ENSURE THAT THE OVERLAY EVELLUM IS THE
 // TOPMOST FORM, LYING ABOVE THE CHILDOVERLAY

  this.SetTopMost(true);
  base.OnMouseUp (e);
}
```

# Appendix III

## Screen Recording to AVI file using Video for Windows Framework (VfW)

In this appendix, we shall present the code that implements the module that records the screen activities in the area chosen by the user to an AVI file. We use the functions defined in the Video for Windows framework to implement this functionality along with the Windows GDI functions that are used to capture the screenshots. The code given in this appendix can be found in the "ScreenCaptureToAVIDll.cpp" file in the "ScreenCaptureToAVIDll" folder of the source code. The "ScreenCaptureToAVIDll.h" contains the declarations and definitions of the member variables. The comments in this code explain the implementation details of the ScreenCaptureToAVIDll module implementation.

```
namespace ScreenCaptureToAVIDll
{
   // THE avi_frame_pointer DEFINES A POINTER TO THE BITMAP //
   REPRESENTING A FRAME IN THE AVI FILE. AN AVI FILE CAN // BE THOUGHT
   OF COMPRISING A SEQUENCE OF BITMAPS.
   LPBITMAPINFOHEADER avi_frame_pointer = NULL;
   // THE avi_streaminfo_header DEFINES THE VARIABLE STORING // THE
   AVISTREAMINFO STRUCTURE. THE STREAM INFORMATION
   // INFORMATION STRUCTURE COMPRISES OF MANY FIELDS
   // THAT STORE STREAM INFORMATION LIKE ITS TYPE (AUDIO/   // VIDEO),
   LANGUAGE, FRAME LENGTH, QUALITY ETC.
   AVISTREAMINFO avi_streaminfo_header;
   // THE avi_file_pointer DEFINES A HANDLE TO THE AVI FILE
   PAVIFILE avi_file_pointer = NULL;
   // THE avi_stream_pointer DEFINES A POINTER TO THE      //
   UNCOMPRESSED STREAM RECORDED TO THE AVI FILE
   PAVISTREAM avi_stream_pointer = NULL;
```

```
// THE compressed_avi_stream_pointer DEFINES A POINTER TO // THE
COMPRESSED STREAM RECORDED IN THE AVI FILE
PAVISTREAM compressed_avi_stream_pointer = NULL;
// THE avi_compression_options DEFINES THE compression  // options
THAT DEFINE HOW THE STREAM MUST BE COMPRESSED // TO OPTIMIZE FOR
SPACE. THE AVICOMPRESSOPTIONS        // STRUCTURE STORES INFORMATION
ABOUT THE COMPRESSOR    // NAME, KEYFRAME RATE, QUALITY OF
COMPRESSION, ETC.
AVICOMPRESSOPTIONS avi_compression_options;


// THE RecordVideo FUNCTION IMPLEMENTS THE RECORDING OF // THE
SCREEN ACTIVITIES IN THE AREA OF INTEREST TO AN  // AVI FILE. THE
DIMENSIONS OF THE CAPTURE AREA AND THE // NAME OF THE AVI FILE ARE
SPECIFIED BY USER INPUTS.
// THE FUNCTION PARAMETERS top, left, width, height
// SPECIFY THE DIMENSIONS OF THE AREA OF CAPTURE. THIS
// IS USUALLY THE DIMENSIONS OF THE DYNAMIC EVELLUM IF
// THE DYNAMIC EVELLUM IS ACTIVE.
// THE aviFileName SPECIFIES THE CHARACTER
// ARRAY CONTAINING THE AVIFILE NAME.
// THE PARAMETER fps SPECIFIES THE FRAME RATE AND THE
// PARAMETER quality SPECIFIES THE QUALITY OF THE AVI
// FILE.


void ScreenCaptureToAVI ::RecordVideo( int top, int left,
          int width, int height, char aviFileName __gc[],
                                    int fps, int quality)
{
    HRESULT hr;
    // szTitle DECLARES A CHARACTER ARRAY OF SIZE BUFSIZE
    // SET TO 255. szTitle STORES THE COMPLETE PATH AND
```

```
// NAME OF THE AVI FILE NAME
char szTitle[BUFSIZE];

// WE SET THE DIMENSIONS OF THE CAPTURE AREA DEFINED
// BY THE MEMBER VARIABLES m_top, m_left, m_width AND
// m_height
m_top=top;
m_left=left;
m_width=width;
m_height=height;

// SET THE m_fps MEMBER VARIABLE THAT DEFINES THE
// FRAME RATE
m_fps=fps;

// SET THE compquality MEMBER VARIABLE THAT DEFINES
// THE VIDEO QUALITY OF THE AVI FILE.
compquality=quality;
// EXTRACT THE NAME OF THE AVI FILE FROM THE
// CHARACTER ARRAY POINTER AND STORE IT AS A
// CHARACTER ARRAY WITH THE NAME szFileName.
szFileName=&aviFileName[0];

// INITIALISE THE avi_frame_pointer TO THE NULL
// POINTER
avi_frame_pointer = NULL;

// INITIALISE THE VARIABLE num_compressor THAT STORES
```

```
// THE NUMBER OF COMPRESSORS TO BE 0.
int num_compressor=0;

// THE CaptureScreenFrame FUNCTION DEFINED IN SCREEN
// -CAPTURETOAVIDLL.CPP, TAKES A SCREEN SHOT OF THE
// USER DESKTOP WINDOW IN THE DIMENSIONS SPECIFIED BY
// THE FUNCTION PARAMETERS & RETURNS A POINTER TO THE
// BITMAP OF THE SCREEN SHOT.
// WE CAPTURE THE FIRST FRAME OF THE AVI FILE IN
// THE DIMENSIONS SPECIFIED BY THE USER.
avi_frame_pointer=
          CaptureScreenFrame(left,top,width,height);

// ALLOCATE 50 BLOCKS OF MEMORY EACH OF THE SIZE OF
// AN ICINFO STRUCTURE TO m_compressor_info.
// m_compressor_info IS A POINTER TO AN ARRAY OF
// ICINFO STRUCTURES. THE ICINFO STRUCTRE STORES
// INFORMATION ON A VIDEO COMPRESSOR. THE ICINFO
// STRUCTURE IS POPULATED WITH VIDEO COMPRESSOR
// INFORMATION WHEN THE VIDEO COMPRESSOR DRIVER
// RECEIVES THE ICM_GETINFO MESSAGE SENT BY THE
// ICINFO FUNCTION.
if( m_compressor_info == NULL )
{
   m_compressor_info =
                (ICINFO*)calloc(50,sizeof(ICINFO));
}
// THE NEXT STEP TO RETRIEVE THE MSU LOSSLESS SCREEN
```

```
// CAPTURE CODEC WHICH IS THE VIDEO COMPRESSOR USED
// TO CREARE A COMPRESSED AVI STREAM.

// WE SEARCH ALL THE INSTALLED VIDEO
// COMPRESSORS ON THE USER'S COMPUTER UNTIL WE FIND
// THE MSU CODEC. THIS IS DONE BY FIRST OPENING A
// VIDEO COMPRESSOR AND THEN RETREIVING ITS
// INFORMATION TO AN ICINFO STRUCTURE. EACH VIDEO
// CODEC IS UNIQUELY IDENTIFIED BY THIS FOURCC CODE.
// THE FOURCC CODE IS STORED IN THE fccHandler FIELD
// OF THE ICINFO STRUCTURE. BY COMPARING THE
// fccHandler VALUE WITH THE FOURCC CODE OF THE MSU
// CODEC WHICH CORRESPONDS TO "scls", WE CAN FETCH A
// HANDLE TO THE MSU SCREEN CODEC.

// WE BEGIN BY POPULATING THE ARRAY OF ICINFO
// STRUCTURES POINTED TO BY m_compressor_info.THIS IS
// DONE BY CALLIN THE ICInfo FUNCTION. THE ICInfo
// FUNCTION RETREIVES THE INCFO STRUCTURE CONTAINING
// COMPRESSOR INFORMATION. THE ICTYPE_VIDEO IN THE
// FOR LOOP BELOW SPECIFIES THAT WE WANT TO ONLY
// CHECK FOR VIDEO COMPRESSORS. A RETURN VALUE OF
// TRUE IMPLIES THAT A VIDEO COMPRESSOR WAS FOUND AND
// THE ICINFO STRUCTURE HAS BEEN POPULATED.

for (int i=0; ICInfo(ICTYPE_VIDEO, i,
     &m_compressor_info[num_compressor]); i++)
```

```
{
   if (i>50) break;
   // IF WE CAN FIND THE MSU CODEC IN THE LIST OF
   // VIDEO COMPRESSORS BY COMPARING THE FOURCC VALUE
   if ( m_compressor_info[num_compressor].fccHandler
               == mimoFOURCC('s','c','l','s'))
   {
        // OPEN THE MSU COMPRESSOR. THE VARIABLE hic
        // REPRENTS A HANDLE TO THE ICINFO STRUCTURE.
        // THE ICOPEN FUNCTION OPENS THE MSU COMPRESSOR
        // IN QUERY MODE TO GET INFORMATION ON THE
        // COMPRESSOR AS SPECIFIED BY THE ICMODE_QUERY
        // ARGUMENT. THE FUNCTION RETURNS A NON-ZERO
        // VALUE IF THE MSU CODEC IS PRESENT.

        HIC hic =
           ICOpen(m_compressor_info[num_compressor].fccType,m_com
           pressor_info[num_compressor].fccHandler,
           ICMODE_QUERY);

        // IF WE HAVE SUCCESSFULLY RETREIVED THE HANDLE
        // TO THE MSU CODEC, WE HAVE TO CHECK FOR ITS
        // COMPRESSION COMPATIBILITY WITH THE FRAME
        // DIMENSIONS SPECIFIED BY THE USER. THIS IS
        // BECAUSE SOME CODECS HAVE SPECIAL
        // SPECIFICATIONS LIKE EVEN DIMENSIONS, ETC.
        // THUS, WE ARE ESSESNTIALLY CHECKING IF THE
        // MSU CODEC CAN COMPRESS THE AVI FRAME WITH
        // USER DEFINED DIMENSIONS TO CREATE A
```

```
// COMPRESSED AVI FILE.
// THE ICCOMPRESSQUERY MACRO QUERIES THE
// COMPRESSOR OF WHICH THE VARIABLE hic
// STORES THE HANDLE TO CHECK IF IT CAN
// COMPRESS THE FRAME STORED AS A BITMAP
// IN THE ADDRESS LOCATION POINTED TO BY THE
// avi_frame_pointer VARIABLE. THE OUTPUT
// FORMAT IS SET TO NULL MEANING THAT THERE IS
// NO SPECIFIC OUTPUT FORMAT WE WANT. THE MACRO
// RETURNS A ICERR_OK IF THE COMPRESSION IS
// SUPPORTED. IT RETURNS A ICERR_BADFORMAT IF
// THE FORMAT IS INCOMPATIBLE WITH THE CODEC.
if (hic)
{
   if( ICERR_OK == ICCompressQuery
             ( hic, avi_frame_pointer, NULL)
     )
   {
         // WE FINALLY POPULATE THE ICINFO
         // STRUCTURE USING THE ICGetInfo
         // FUNCTION.
         ICGetInfo( hic,
         &m_compressor_info[num_compressor],
         sizeof(ICINFO));

         // WE SAVE THE INDEX OF MSU CODEC
         // IN THE selected_compressor VARIABLE
         selected_compressor=num_compressor;
```

```
                    // WE STORE THE FOURCC CODE IN THE
                    // MEMBER VARIABLE compfccHandler
                    compfccHandler =
                    m_compressor_info[selected_compressor].
                    fccHandler;

                    // WE CLOSE THE COMPRESSOR AND BREAK
                    // OUT FROM THE LOOP
                    ICClose(hic);
                    break;
                }
            }
        }
}
// NOW THAT WE HAVE SUCCESSFULLY RETRIEVED THE
// MSU CODEC, WE HAVE TO INITIALISE THE AVIFILE LIBRARY
// INORDER TO CREATE AN AVI FILE AND START WRITING THE
// STREAM TO IT. THE AVIFILEINIT FUNCTION PERFORMS THE
// INITIALISATION.
AVIFileInit();
// OPEN THE AVI FILE FOR WRITING. IF THE AVI FILE DOES
// NOT EXIST, WE CREATE IT. THE avi_file_pointer STORES
// THE POINTER TO THE AVIFILE INTERFACE. THE AVIFILEOPEN
// FUNCTION OPENS AN AVI FILE WITH A USER DEFINED NAME
// AND GIVES THE HANDLE TO THE AVIFILE INTERFACE IF
// SUCESSFUL. THE szFileName SPECIFIES THE NAME OF THE
// AVI FILE WITH THE COMPLETE PATH.
```

```
hr =   AVIFileOpen (&avi_file_pointer, static_cast<LPC
      TSTR>(const_cast <void*>(static_cast<const void*>
      (System::Runtime::InteropServices::Marshal::
      StringToHGlobalAnsi(szFileName)))),
      OF_WRITE | OF_CREATE | OF_SHARE_EXCLUSIVE, NULL);

// ONCE THE AVIFILE IS CREATED, WE HAVE TO FILL IN THE
// DETAILS OF THE AVI STREAM. THE AVISTREAMINFO
// STRUCTURE STORES THE DETAILS OF THE STREAM
// INFORMATION. THE STEAM TYPE IS SPECIFIED BY THE
// fccType FIELD AND IS SET TO streamtypeVIDEO
// INDICATING THAT WE ARE GOING TO WRITE A VIDEO STREAM
// ONLY.THE FRAMERATE IS SPECIFIED BY THE dwRate FIELD.
// THE fccHandler SPECIFIES THE FOURCC CODE OF THE C
// COMPRESSOR THAT IS GOING TO COMPRESS THE STREAM WHILE
// IT IS BEING SAVED TO THE AVIFILE. IN OURCASE, IT IS
// THE MSU CODEC. The dwScale and dwRate specify the
// frame rate in case of video streams.
// dwSuggestedBufferSize STORES THE BUFFER SIZE OF THE
// STREAM NEEDED FOR PLAYBACK. SINCE EACH CHUNK OF THE
// AVI FILE SHALL BE OF THE SIZE OF THE BITMAP THAT
// STORES THE AVI FRAME WITH DIMENSIONS SPECIFIED BY THE
// USER.
memset(&avi_streaminfo_header, 0,
           sizeof(avi_streaminfo_header));
avi_streaminfo_header.fccType     = streamtypeVIDEO;
avi_streaminfo_header.fccHandler = compfccHandler;
avi_streaminfo_header.dwScale = fps;
```

```
avi_streaminfo_header.dwRate= fps;
avi_streaminfo_header.dwSuggestedBufferSize  =
                          avi_frame_pointer->biSizeImage;

// WE THE SET THE DIMENSIONS OF THE FRAME IN THE AVIFILE
// THE  rcFrame IS THE RECTANGLE DEFINING THE FRAME
// DIMENSIONS
SetRect(&avi_streaminfo_header.rcFrame, 0, 0,
(int) avi_frame_pointer->biWidth,
          (int) avi_frame_pointer->biHeight);

// WE CREATE A UNCOMPRESSED VIDEO STREAM ASSOCIATED WITH
// THE CREATED AVIFILE HAVING PROPERTIES SPECIFIED BY
// THE avi_streaminfo_header STRUCTURE.
// THE AVIFileCreateStream CREATES A VIDEOSTREAM AND
// RETURNS A POINTER TO THE STREAM INTERFACE. WE THEN
// CREATE A COMPRESSED STREAM FROM THIS AVI STREAM.
hr =   AVIFileCreateStream(avi_file_pointer,
                           &avi_stream_pointer,
                           &avi_streaminfo_header);

// WE SPECIFY OUR COMPRESSION OPTIONS IN THE
// AVICOMPRESSOPTIONS STRUCTURE. THE AVICOMPRESSOPTIONS
// STRUCTURE SPECIFIES HOW THE AVI STREAM HAS TO BE
// COMPRESSED BEFORE WRITING TO THE AVI FILE. THE FIELD
// dwKeyFrameEvery SPECIFIES THE KEYFRAME RATE. SINCE
// WE TREAT EACH FRAME INDEPENDENT OF THE OTHER, THE
// dwKeyFrameEvery HAS THE SAME VALUE AS THAT OF THE
// FRAME RATE OR fps. THE dwQuality SPECIFIES THE VIDEO
```

```
// QUALITY AND ITS VALUE IS USER DEFINED. THE
// dwBytesPerSecond SPECIFIES THE DATA RATE IN BYTES.
// SINCE WE DO NOT KNOW THE DATA RATE, WE SET IT AS 0.
// THE lpFormat FIELD SPECIFIES THE POINTER TO THE
// STRUCTURE THAT DEFINES THE OUTPUT FORMAT OF THE COM-
// PRESSED FRAME. SINCE WE DO NOT HAVE A SPECIFIC OUTPUT
// FORMAT, WE SET THIS TO 0X0. THE cbFormat SPECIFIES
// THE SIZE IN BYTES OF THE lpFormat. THIS IS SET TO 0
// SINCE WE DO NOT HAVE THE VALUES. THE
// dwInterleaveEvery IS NOT VALID IN OUR CASE SINCE WE
// ARE NOT INTERLEAVING THE UNCOMPRESSED STREAM WITH THE
// COMPRESSED STREAM WHILE WRITING IT TO THE FILE. THE
// dwInterleaveEvery IS SET TO 0.
memset(&avi_compression_options, 0,
    sizeof(avi_compression_options));
avi_compression_options[0]->fccType =  streamtypeVIDEO;

avi_compression_options[0]->fccHandler = compfccHandler;
avi_compression_options[0]->dwKeyFrameEvery = fps;
avi_compression_options[0]->dwQuality = compquality;
avi_compression_options[0]->dwBytesPerSecond = 0;
avi_compression_options[0]->dwFlags = AVICOMPRESSF_VALID
                                |AVICOMPRESSF_KEYFRAMES;
avi_compression_options[0]->lpFormat = 0x0;
avi_compression_options[0]->cbFormat    = 0;
avi_compression_options[0]->dwInterleaveEvery = 0;

// NOW THAT WE HAVE THE COMPRESSION OPTIONS SPECIFIED,
```

```
// WE CREATE THE COMPRESSED STREAM FROM THE UNCOMPRESSED
// STREAM SPECIFIED BY avi_stream_pointer AND THE
// avi_compression_options. THE compressed_avi_stream_-
// -pointer STORES THE HANDLE TO THE POINTER POINTING
// TO AVI STREAM INTERFACE.
hr =  AVIMakeCompressedStream
    (&compressed_avi_stream_pointer, avi_stream_pointer,
     &avi_compression_options, NULL);

// WE SPECIFY THE FORMAT OF THE STREAM USING THE
// AVIStreamSetFormat FUNCTION.
hr = AVIStreamSetFormat(compressed_avi_stream_pointer,
     0, avi_frame_pointer,       // stream format
     avi_frame_pointer->biSize +   // format size
     avi_frame_pointer->biClrUsed * sizeof(RGBQUAD));

// SINCE WE NO LONGER NEED THE UNCOMPRESSED AVI FRAME,
// WE DELETE IT AND RELEASE THE USED MEMORY SPACE.
FreeFrame(avi_frame_pointer);

// THE COMPRESSED FRAMES ARE WRITTEN TO THE AVI FILE IN
// A WHILE LOOP AS LONG AS THE recordstate IS TRUE.
// THE RECORDSTATE IS A PUBLIC MEMBER OF THE CLASS AND
// CAN BE SET OR RESET FROM THE CAPTURECONTROLLER CODE.
// IN THE EVENT THE USER STOPS A SCREEN RECORDING, THIS
// VARIABLE IS SET TO 0 AND WE BREAK FOM THE WHILE LOOP
// AND SAVE THE AVI FILE.
```

```
while (recordstate) // WHILE WE ARE IN RECORDMODE LOOP
{
  frametime = 0;
  // CAPTURE THE USER DEFINED AVI FRAME
  avi_frame_pointer = CaptureScreenFrame (
                          m_left,m_top,width, height);

  // GET THE CURRENT TIME
  frametime = GetCurrentTime();

  // RECHECK IF ALL THE POINTERS ARE VALID AND THE
  // RECORD STATE IS TRUE
  if((compressed_avi_stream_pointer) &&
     (avi_stream_pointer) && (avi_file_pointer) &&
    (recordstate)&& (avi_frame_pointer))
  {
       // THE AVIStreamWrite WRITES THE FRAME TO THE
       // COMPRESSED AVI STREAM AFTER COMPRESSING IT
       hr = AVIStreamWrite
            (compressed_avi_stream_pointer,
             frametime, 1,
             (LPBYTE) avi_frame_pointer +
             avi_frame_pointer->Size +
            avi_frame_pointer->biClrUsed *
            sizeof(RGBQUAD),
             avi_frame_pointer->biSizeImage,
             0, NULL, NULL);
  }
  // ONCE WE HAVE WRIITEN THE FRAME, FREE THE MEMORY.
```

```
      FreeFrame(avi_frame_pointer);
     } // END OF WHILE LOOP
      // IF THE USER RESETS THE RECORD STATE TO FALSE AS IN
      // THE CASE OF AN EVELLUM RESIZE, WE STOP WRITING
      // FRAMES TO THE AVI FILE AND GO TO THE
      // StopRecordingVideo FUNCTION THAT CLOSES AND SAVES
      // THE AVI FILE.
      StopRecordingVideo();
      return ;
   }
}

// THE StopRecordingVideo FUNCTION STOPS THE RECORDING
// PROCESS, CLOSES THE AVI FILE AND FREES ALL THE MEMORY
void ScreenCaptureToAVI::StopRecordingVideo()
{
      HRESULT hr;
      recordstate=0;
      // FREE THE MEMORY SPACE STORING THE VIDEO
      // CODEC INFORMATION
      free( ScreenCaptureToAVIDll::
            ScreenCaptureToAVI::m_compressor_info);

      // WE CLOSE THE COMPRESSED STREAM AND RELEASE THE
      // COMPRESSED STREAM POINTER
      if (compressed_avi_stream_pointer)
      {
```

```
hr=AVIStreamClose(compressed_avi_stream_pointer);
compressed_avi_stream_pointer=NULL;
}
// WE CLOSE THE UNCOMPRESSED STREAM AND RELEASE THE
// UNCOMPRESSED STREAM POINTER
if (avi_stream_pointer)
{
hr=AVIStreamClose(avi_stream_pointer);
avi_stream_pointer=NULL;
}
// WE FINALLY CLOSE THE AVI FILE AND RELEASE THE FILE
// POINTER
if (avi_file_pointer)
{
hr=AVIFileRelease(avi_file_pointer);
avi_file_pointer=NULL;
}
// SINCE WE CALLED AVIFILEINIT() ONCE, WE RELEASE THE
// LIBRARY MANUALLY WITHOUT FAIL.
AVIFileExit();

// IN ORDER TO AVOID ANY DANGLING POINTERS, WE SET ALL
// THE POINTER VALUES TO NULL.
avi_frame_pointer=NULL;
first_avi_frame_pointer=NULL;
memset(&avi_streaminfo_header, 0,
        sizeof(avi_streaminfo_header));
```

```
      memset(&avi_compression_options, 0,
              sizeof(avi_compression_options));
} //END OF STOPRECORDINGVIDEO FUNCTION


// THE CaptureScreenFrame FUNCTION DEFINED IN THE
// ScreenCaptureToAVIDll.cpp  RETURNS THE BITMAP OF THE
// SCREEN AREA DEFINED BY THE USER. THE BITMAP IS A FRAME
// WRITTEN TO THE AVI FILE AFTER COMPRESSING IT TO OPTIMISE
// FOR SPACE.THE FUNCTION RETURNS A HANDLE TO THE
// BITMAPINFOHEADER POINTER. THE BITMAPINFOHEADER STORES
// THE BITMAP AS WELL AS ITS ASSOCIATED DETAILS LIKE
// DIMENSIONS, ETC. WE USE THE WINDOWS GDI FUNCTIONS
// TO ACCOMPLISH THIS.
LPBITMAPINFOHEADER ScreenCaptureToAVI:: CaptureScreenFrame
                   (int left,int top,int width, int height)
{

           // WE CREATE 2 BITMAPS. ONE TO STORE THE COMPLETE
           // BITMAP OF THE SCREEN AND THE OTHER TO STORE
           // THE ACTUAL BITMAP OF THE FRAME.WE TECHNICALLY
           // TAKE THE BITMAP OF THE SCREEN AND COPY THE
           // ONLY THE DESIRED SCREEN AREA TO THE NEW BITMAP
           // HERE, THE VARIABLE hbm HOLDS THE NEW BITMAP
           // WHILE THE oldbm VARIABLE HOLDS THE BITMAP
           // OF THE SCREEN/DESKTOP WINDOW.
           BITMAP hbm,oldbm;

           // GET THE DEVICE CONTEXT OF THE DESKTOP WINDOW.
```

```
// A DEVICE CONTEXT IS A CONTAINER ON WHICH YOU
// YOU CAN DRAW. THE HDC DATA TYPE STORES THE
// HANDLE TO A DEVICE CONTEXT(DC). THE DC CAN
// BE ANY WINDOW ON THE SCREEN, THE WHOLE SCREEN
// OR JUST A PLAIN BITMAP FILE. WINDOWS DOES NOT
// PERMIT THE USER TO DIRECTLY ACCESS THE CONTENT
// OF A DC. THE USER CAN USE OTHER GDI FUNCTIONS
// PERFORM OPERATIONS ON IT LIKE, GET ITS DATA,
// COPY ITS DATA TO ANOTHER BITMAP, ETC.
// SINCE WE WANT A SCREENSHOT OF AN AREA OF THE
// SCREEN, WE FIRST NEED TO GET ITS DC. THE
// GetDC FUNCTION GETS THE HANDLE TO THE DC OF
// A HANDLE REPRESENTING A WNDOW. HERE, WE PASS
// THE   GetDesktopWindow FUNCTION THAT RETUNS
// THE HANDLE TO THE DESKTOP WINDOW IN ORDER TO
// RETREIVE THE DC OF THE SCREEN.
HDC hScreenDC = ::GetDC(GetDesktopWindow());

// CREATE DOUBLE BUFFER OF THE SAME SIZE
HDC hMemDC = ::CreateCompatibleDC(hScreenDC);

//CREATE AN EMPTY BITMAP OF USER DEFINED SIZE
hbm = CreateCompatibleBitmap
                (hScreenDC, width, height);

// CREATE BITMAP OF WHOLE SCREEN
oldbm = (HBITMAP)SelectObject(hMemDC, hbm);

// SET THE COPY FLAGS. SRCCOPY FLAG INDICATES
```

```
// TO THE BITBLT FUNCTION TO DIRECTLY COPY THE
// SOURCE BITMAP RECTANGLE TO THE DESTINATION NEW
// BITMAP. THE CAPTUREBLT FLAG IS NECESSARY SINCE
// WE NEED TO CAPTURE TRANSPARENT, SEMI
// TRANSPARENT AS WELL AS LAYERED WINDOWS.
// SINCE THE DYNAMIC EVELLUM IS COMPOSED OF A
// TRANSPARENT AND SEMI-TRANSPARENT WINDOW
// LAYERED ON ONE ANOTHER, THE USE OF THE
// CAPTUREBLT FLAG IS AN ABSOLUTE NECESSITY.
DWORD bltFlags = SRCCOPY;
bltFlags |= CAPTUREBLT;

// COPY THE DESIRED AREA OF CAPTURE TO THE
// NEW BITMAP. THE BitBlt FUNCTION ACCOMP-
// LISHES A BIT BY BIT COPYING OF THE BITS
// IN THE DESIRED AREA OF THE SCREEN BITMAP
// TO THE NEW BITMAP.
BitBlt( hMemDC, 0, 0, width, height,
          hScreenDC, left, top, bltFlags);

SelectObject(hMemDC,oldbm);

// CREATE THE BITMAPINFOHEADER USING THE DIB
// (DEVICE INDEPENDENT BITMAP) CREATED FROM THE
// BITMAP. THE BitmaptoDIB FUNCTION CREATES A
// DIB FOR THE GIVEN BITMAP.
LPBITMAPINFOHEADER pBM_HEADER =
 (LPBITMAPINFOHEADER)GlobalLock(BitmaptoDIB(hbm,
                                    m_bits));
```

```
            if (pBM_HEADER == NULL)
            {
                  // WE PRINT AN ERROR MESSAGE IF WE FAIL TO
                  // CREATE THE BITMAPINFOHEADER
                  MessageBox(NULL,"Error reading a
                        frame!","Error",MB_OK |
                              MB_ICONEXCLAMATION);

                  //FREE ALL MEMORY
                  DeleteObject(oldbm);
                  DeleteObject(hbm);
                  DeleteDC(hScreenDC);
                  DeleteDC(hMemDC);
                  exit(1);
            }
            //FREE ALL MEMORY
            DeleteObject(oldbm);
            DeleteObject(hbm);
            DeleteDC(hScreenDC);
            DeleteDC(hMemDC);

            // RETURN THE BITMAPINFOHEADER WHICH HAS
            // INFORMATION ABOUT THE BITMAP AS WELL THE
           // BITMAP
            return pBM_HEADER;
   }

}//END OF SCREENCAPTURETOAVIDLL NAMESPACE
```