WriteOn – A Tool for Effective Classroom Presentations

Vinod Eligeti

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

Dr. Joseph G. Tront, Chair
Dr. Osman Balci
Dr. Stephen Edwards

Dec 16, 2005
Blacksburg, Virginia

Keywords: Tablet PC, Electronic ink, Classroom presentation, Computer-aided instruction

WriteOn – A Tool for Effective Classroom Presentations

Vinod Eligeti

Abstract

This thesis provides an introduction to an advance in technology-aided instruction. Most of the research in this area has focused on PowerPoint® based applications or white board-centered electronic ink applications with the capability of broadcasting slides, ink annotations and so forth, used for presentation or classroom lectures. But these tools lack the capability of annotating on any kind of applications with active content playing (a movie or a simulation, for instance) in the background. Additional useful, but currently unavailable functionality would include the capability of broadcasting the presentation information, which can further consist of lecture slides, ink annotations, video of the desktop screen activity, or any other application program that might be used to demonstrate a concept or illuminate an idea. Therefore, the current research attempts to provide these facilities with a new tool, *WriteOn*. WriteOn improves both the presentation of information and the interactivity in classroom instruction, because it gives the instructor the ability to ink annotate on any application by using a virtual transparency surface, called *electronic vellum* or simply *eVellum*, which in effect resides on top of all desktop window applications. The instructor can enable the vellum at any point during the lecture and write on it to draw diagrams, make notes, emphasize points, or otherwise elucidate the presentation content. The instructor can also pierce the eVellum in order to switch to different applications, modify an applications parameters or operating values, or otherwise manipulate an operating program as part of a classroom demonstration or discussion. These features allow the instructor to demonstrate the dynamic operation of any application, which is an improvement on a static PowerPoint display of a program's operation. With WriteOn, the instructor can save the ink annotations along with desktop screen activity over an interval of time as a movie file and

later make this file available to students. Alternatively, the instructor can transmit to the students the presentation information along with ink annotations in real-time so that the students can make their own notes on top of information being produced by the instructor. Thus the tool can be used to enhance the interactive lecturing process and help students to develop good note-taking processes and habits.

WriteOn is also capable of saving the voice of the instructor, provided there is an audio device attached to the instructor’s Tablet PC. However, broadcasting the instructor’s voice is not yet fully supported. The WriteOn tool was developed using Microsoft’s technologies: Windows Media Encoder® and DirectShow of DirectX®, as well as Microsoft’s ConferenceXP API to achieve streaming of the presentation information.

The first chapter explains the need for computer tools used for effective teaching purposes. The second chapter presents the architectural and technical details of WriteOn. Chapter three describes the architecture of the WriteOn tool. Chapters four through six explain the major software components of the system and also give the pros and cons of the DirectShow and Windows Media Encoder technologies. The seventh chapter provides an explanation of the usage of the tool by instructors and students. The eighth chapter presents the experiences of the instructors and students using the WriteOn tool in the classroom and concludes with a discussion of future work in this area. The Appendix V provides a developers guide for those who might like to expand on this open source code.

Acknowledgments

The research thesis would not have been possible without the support of the many people who directly or indirectly helped me to realize the WriteOn tool. First of all I wish to express gratitude to my advisor Dr. Joseph G. Tront. His never ending support, encouragement, guidance and patience from the very beginning have been invaluable to me in the difficult task of completing my research. Appreciation is also extended to other thesis committee members, Dr. Osman Balci and Dr. Stephen Edwards, for their valuable advice and comments on strengthening this work.

I also appreciate the help from the ConferenceXP team of Microsoft and several technical newsgroups in fostering my understanding of the technologies used in this work. Their expert technical advice reduced the development time and eased the effort required to build the system which is the result of my research. I am grateful for their patience with and prompt responses to questions related to my work. I would also like to thank Michael Potter for conducting some of the usability tests and Graphical User Interface suggestions about the system.

Finally, I express my deepest appreciation and gratefulness to my mother and father, Shashikala and Rajanarendar Eligeti, who have always provided me with the encouragement to acquire the education I wanted. They were always there for me when I needed them and I cannot imagine myself going through all this without their help and support.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Effective lecturing is a daunting task for instructors attempting to deliver knowledge to their students. Instructors, therefore, use various classroom aids to make the lecture more interesting as they try to communicate important points about the subject without much deviation from the subject. Some of these include the blackboard in a classroom, a projector for animated display, hypermedia and so forth. Use of a blackboard, the most common instructional technique, provides the instructor as well as the students with a good amount of writing space. The primary advantage of the classroom blackboard is that it gives a comprehensive view of all the writings on it, thus helping students grasp the subject matter more easily. Instructors also use projectors and hypermedia content to produce an effective teaching process. These teaching aids are especially useful for the teachers of history, geography and other Liberal Arts courses. But these teaching aids do not give the instructor the capability of easily saving the lecture information or students the capability of annotating (note-taking) the lecture while the instructor is delivering it. Nor do these tools allow students to easily capture dynamic information that is added to a lecture during an active classroom presentation. Student annotations (notes) of the lecture information is an important factor for effective learning processes as they help the student to assimilate the presentation more easily and review the lecture to gain a better understanding of the material. Computer-assisted teaching/learning provides a viable solution for some of these problems.

Computer-Aided instruction (or Computer-Assisted instruction) can be readily used to enhance the teaching/learning process, because there are several available tools

(PowerPoint, WordPad and Notepad) which require minimal effort from the user to learn them. Rune Kornefors and Lennart Lundberg explain the importance of computers for effective teaching [1]. The predominant tool used by instructors for classroom presentations is Microsoft's PowerPoint. Using this tool an instructor can build animated slides which help explain the nature of the subject matter and can be somewhat dynamic. Moreover, using the tool gives a structured approach to the lecture. The PowerPoint approach, however, does not provide students with built-in note-taking capabilities. Students often have difficulty taking notes using tools like Notepad® and WordPad® because this kind of note-taking interrupts their concentration on the lecture. Several companies including Toshiba, HP, Sony, Microsoft and others have, therefore, studied classroom scenarios and introduced products such as Tablet PCs, computers with the capability of annotating on the screen, and Notebooks – to provide portable, but less robust note taking platforms. Tools such as OneNote®, Journal® and Classroom Presenter® (developed by faculty and students at the University of Washington) use the Tablet PC to make teaching more productive. The ability to ink on lecture information using these applications has replaced the conventional note-taking methods. Steven A. Wolfman [2] has detailed several advantages of the Classroom Presenter tool in a classroom scenario. Michelle Wilkerson, William G. Griswold and Beth Simon [3] elaborate on use of Classroom Presenter in a virtual classroom. These studies make it clear that the usage of Tablet PC-based tools enhances the learning process because of features that include: instructor inking on the lecture slides, student submissions of an annotated slide for class discussion or instructor assessment, and the ability for students to save the presentation and their associated lecture notes made directly on a localized copy of the presenters information.

1.2 The problem and the solution

Computer-Assisted instruction has become an important topic among educational researchers interested in building computer application tools for improving classroom presentations. Using ink-based tools can augment the interaction between the students and their instructor in a physical classroom as well as in a virtual classroom. Because such tools allow knowledge to be more easily shared, their use enriches the classroom experience for students. Most of the tools developed for classroom presentation, as an outcome of intense research; attempt to provide a better solution for enhancing active lecturing. However these tools have failed to implement some of the important elements of the computer-assisted teaching and learning process. These include: the extemporaneous e-ink annotations on any application running on a Tablet PC, broadcast of the instructor's presentation information possibly as slides of a PowerPoint-based applications or desktop screen activity, or recoding of the voice of the instructor. The research work presented in this thesis realizes the above features in the form of a software tool called *WriteOn*. *WriteOn* is a computer-assisted tool which enhances the classroom experience for the students and the instructor.

1.2.1 Anywhere ink annotations

Annotating using an electronic ink stylus allows an instructor to illustrate and elucidate the lecture topic by inking on the application used for presentation in a natural manner, similar to the way in which an instructor would use a chalkboard or ink pen and plastic transparency material. The capability to apply electronic ink anywhere on the screen gives instructors the option of using more than one kind of application for presentations, thereby enhancing the effectiveness of their lectures. Classroom presentation tools such as PowerPoint cannot be effectively used to demonstrate the behavior of operating programs. For example, when an instructor is required to explain the use of MicroChip®, an Integrated Development Environment (IDE) used for assembly language development and simulation, it is much more effective to annotate on top of an actively operating application rather than simply capturing the static screen shots of MicroChip® simulation, inserting them into PowerPoint slides, and explaining concepts from the fixed images. The static PowerPoint approach clearly does not effectively illustrate the way MicroChip simulation works. Students at the same time cannot easily begin to comprehend the whole assembly and simulation process based on a static presentation. They must have an opportunity to pose the "what-if" questions in class and have the instructor address them by both demonstrating the application operation and by making notes on the functionality of the application. Lacking the ability to do in class interactive exploration guided by the instructor, the student must start from the beginning of the conceptual presentation and run the application after the classroom session, which likely leads to a less efficient use of their time. Further time savings occur through the use of delivery-time ink annotation on applications since the instructor need not prepare PowerPoint slides of an application demonstration. Also by saving the ink annotations made on the running applications, students and the instructor save time in taking notes rather than hand-writing on paper, or typing notes in a WordPad® or a Notepad®, which can be slow and inaccurate. Because note taking in WordPad® or Notepad® does not give a base context of the presentation, students have to spend a considerable amount of time transcribing drawings and text from presentation screen or blackboard to their note-taking mechanism, which causes them to lag behind the lecture lose concentration on the subject matter of the classroom presentation.

1.2.2 Broadcast of presentation information

In order for students to view the presentation in real-time on their computer and to ink annotate on it at the same time, it is necessary to broadcast the presentation information within the learning environment. The information broadcast may take the form of a continuous movie of the instructor's screen activity, screen shots of the presenters Tablet PC/Laptop screen, the voice of the instructor, or any other information that supports transferring information to the student. Broadcast or streaming of presenter information also provides the capability of connecting to the classroom presentation remotely. By doing so, students will be in the same context of the lecture as that of the instructor. It also helps remotely located students to take notes based on the presentation directly displayed on their machines.

Based on the above mentioned features, WriteOn attempts to be an effective teaching/learning support tool. It is built using Microsoft Ink technology for ink annotation; DirectShow and the ConferenceXP Software Developer Kit (SDK) are used for broadcasting the ink and desktop screen activity. Windows Media Encoder is also used to achieve some of the requirements which will be discussed in the later sections. The details of the integration of the above mentioned software technologies to create the WriteOn tool is described in Chapter 4. The next chapter describes some of the related work on the design and use of tablet-technology teaching/learning tools performed by other researcher.

Chapter 2

Related work

This chapter presents the significant past and present research on tools in the field of active-lecturing. Some of these tools include Classroom Presenter, Virtual Network Computing and ink tools such as Snipping Tool, DyKnow and others.

2.1 Overview of the Classroom Presenter tool

Classroom Presenter (or simply *Presenter*), the result of a Computer Science [10] project, at the University of Washington has conducted in collaboration with the Microsoft Research group, facilitates an active lecture by allowing for student submissions and their feedback. Presenter is a PowerPoint-centered application where the instructor and the students can ink on static presentation slides that are pre-made prior to presentation. Students may save their locally annotated version of the presentation as class notes for review outside the classroom in very much the same way as standard pencil and paper notes are reviewed. To communicate with one another, the instructor and the students connect to a virtual electronic classroom venue through an Ethernet network or Wireless network with multicast enabled. The instructor next opens a file that contains the PowerPoint slides formatted for use in Presenter. The instructor needs to prepare the presentation file in the CSD or RTP file format (also called the Deck). To create a CSD or RTP file, the program called DeckBuilder is used. DeckBuilder is a computer tool which is shipped along with the Classroom Presenter program. Given a PowerPoint file, the DeckBuilder converts it to a CSD or RTP format file. Another way to create the CSD or a RTP file is to use a separately available plug-in for PowerPoint. Once the plug-in is installed in PowerPoint, the instructor need only use the command Export to CSD to transform the PowerPoint presentation file to a .csd or .rtp file. The

PowerPoint plug-in, [10] has the link to download and install this plug-in, also allows the instructor to generate objects in the PowerPoint slides that when converted to Presenter CSD format, will only show up in the instructor’s view. These are referred to as instructor’s notes and are very useful prompts during lectures.

Students who are connected to the instructor’s machine via the virtual classroom created by Presenter can request that the slides be downloaded to their machines. Instructors have the capability to allow the students to navigate freely through the entire set of slides for the presentation or to be strictly linked to showing just the slide visible on the instructor’s presentation system. This capability allows the instructor to control student classroom behaviors and is particularly useful for lower division students who may not have the self-discipline to pay close attention to the classroom discussion.

Students can ink annotate on the instructor slides broadcast to them, submit their annotations back to the instructor for classroom discussion. This type of interaction is called Student Submissions. These annotations can take the form of comments or questions or explanations of the subject matter by the students. The instructor must enable Student Submissions to allow the students to select their ink annotations and send them to the instructor system. Using this mechanism, the instructor can pose exercise questions to the class, have the students solve them, and then discuss the solutions with the rest of the class. In this way, the instructor can engage the students in the lecture as well as better gauge the progress of the class in understanding the lecture material.

Students can also give real-time feedback to the instructor. Feedback takes the form of a message sent by the student to the instructor’s machine. The message will be displayed on the instructor’s machine in the screen location specified by the student. For example, if the instructor is presenting the lecture too fast, the students can choose to send the “Go Slow” message from the feedback menu. Or if any part of the presentation is not clear, s/he can send “More Explanation” the feedback message. Once the student sends the feedback message, a dot for each message appears on the presentation slide of the instructor. This helps the instructor to how to better proceed with the lecture presentation. The feedback message does not appear on the projector presentation though.

The Classroom Presenter software provides the facility to save the lecture presentation session as a movie file which can be only played by the Classroom Presenter Playback tool. Students can review classroom note activity by replaying the ink strokes as they are applied to the lecture material. Unfortunately, Classroom Presenter does not currently record audio to play back along with ink strokes.

Richard Anderson, Ruth Anderson, Crystal Hoyer, Beth Simon, Fred Videon, and Steve Wolfman detail the use of Classroom Presenter for lecturing purpose in [4]. Beth Simon, Ruth Anderson, Crystal Hoyer and Jonathan Su summarized their experiences of use of tablet PC-based system to achieve active learning in computer sciences courses in [12]. The experience is strengthened by the paper [13] authored by Richard Anderson, Ruth Anderson, Beth Simon, Steve Wolfman, Tammy VanDeGrift and Ken Yasuhara. In [14], Richard Anderson, Ruth Anderson, Crystal Hoyer and Steve Wolfman gives an insight into the use of digital ink in the lecture presentations. The above mentioned papers further establish the fact that the use of tablet PC and electronic ink provide and improve the active learning in a classroom. This is one of the motivations in building the WriteOn tool.

But the Presenter tool cannot annotate on any other applications except on its own slides as derived from PowerPoint. This greatly reduces the instructor’s flexibility to annotate on the demonstration of operating software tools and other applications which are important to classroom discussions. Also, for the most part, it is not possible to introduce the dynamic functionalities of other applications into PowerPoint slides for presentation.

2.2 Other remote desktop tools

Virtual Network Computing (*VNC*) is another remote desktop tool, [5] is the official website on VNC tool that is sometimes used in classroom environments. Basically, VNC is a remote desktop for users to control a remote machine rather than using for an active lecturing process. Using this tool, the instructor can start a presentation on his/her machine and the students can connect to the instructor’s machine to view the presentation. The instructor can run any other ink-based applications to ink on the presentation slides. Any mouse events, keyboard events, etc. in the students machine also generates the same events in the instructor machine. With this virtual connection to the instructor’s machine, students can annotate on the instructor’s presentation. This situation opens possibilities for interaction during lectures, but it also poses some problems. In this scenario, the instructor’s presentation slides get modified in a manner not under instructor control or scrutiny. Giving entire control of the instructor’s presentation system to the students is generally undesirable. Since VNC actually does not allow ink-annotations, the user has to employ other tools for this purpose. Also student systems make a unicast connection to the instructor’s remote machine instead of a multicast connection. A significant amount of network traffic occurs in this situation when large numbers of images are passed over the network. Network congestion makes this configuration less than optimal when large classes are involved.

DyKnow is another computer-aided teaching tool. [11] web-site gives the product information in detail. DyKnow allows teachers to keep students engaged in learning. It reduces time in preparing the presentation through use of existent content like PowerPoint, web pages, image files, etc. It also allows the instructor to get quick feedback from the students. Since as an electronic ink application tool, the instructor and the students can take notes by annotating while lecture presentation is going on. Though DyKnow has good features yet it is not capable of annotating on the desktop with other applications running. This deprives the instructor of annotating on any dynamic application windows.

Microsoft’s OneNote also falls under the electronic ink based application tools category. OneNote is an excellent tool for note taking. OneNote brings up a notebook page type of a window where students can annotate on the window. Features such as saving the images in OneNote, recording the voice of the students or the instructor, allowing the students to create multiple pages to annotate on and others enrich the user experience in a classroom environment. Instructors or students can share the OneNote files through Email or by starting a session or by joining an already created session. Thus OneNote provides a great note taking capabilities for students. However, it is not as extensible as DyKnow and cannot annotate on a dynamic application window and then broadcast at the same time. For further information about OneNote and its usage please refer [12].

Lastly under ink tools, Snipping Tool can annotate and capture the image of the ink annotation on the desktop. Snipping tool comes with the installation of Microsoft Experience Pack for tablet PCs. [16] gives the product information about the Experience Pack and the Snipping tool. The Snipping Tool basically takes a snapshot picture of the background image of a selected portion of the screen and displays a transparent film over it. Users can annotate on this film. Snipping Tool then captures only the bounding image of the first stroke (the selection stroke) and copies all the bitmap information of the background. Ink strokes made after the selection stroke are just annotations on the snipped image. Instructors can embed these image files into any application that accepts objects from the operating system clipboard. For example, DyKnow can include the snapshots taken from the Snipping tool for lecture presentations. Since the Snipping Tool takes a snapshot, i.e., a single frame in time, the user cannot annotate on dynamic output or video application windows. Also, the Snipping Tool does not directly allow broadcast of the snipped image thus limiting its usefulness for active lecturing purposes.

The WriteOn tool attempts to incorporate the beneficial features of the tools mentioned above while at the same time overcoming the difficulties to provide an easy to use and viable solution that will enrich and enliven classroom presentations.

Chapter 3

System Overview

This section explains the architecture of the WriteOn tool. WriteOn is written in the C# language of the Microsoft .NET framework. WriteOn is designed under the well known Model View Controller (MVC) paradigm and is used to describe throughout the chapter. It is used for classroom presentations and designed to work primarily on a Tablet PC.

3.1 The .Net Framework

The .Net Framework is a Microsoft technology which helps in building applications that can run on any Windows Operating Systems if the OS has the .NET runtime engine installed. By using .NET, the developer does not need to rebuild the application to run in each of the several different Windows Operating Systems. The above feature is also called compile once and run anywhere. This is basically achieved by generating an intermediate code after compilation, which is then being interpreted later by the Common Language Runtime (CLR) of the .NET Framework. This is shown in Figure 1.

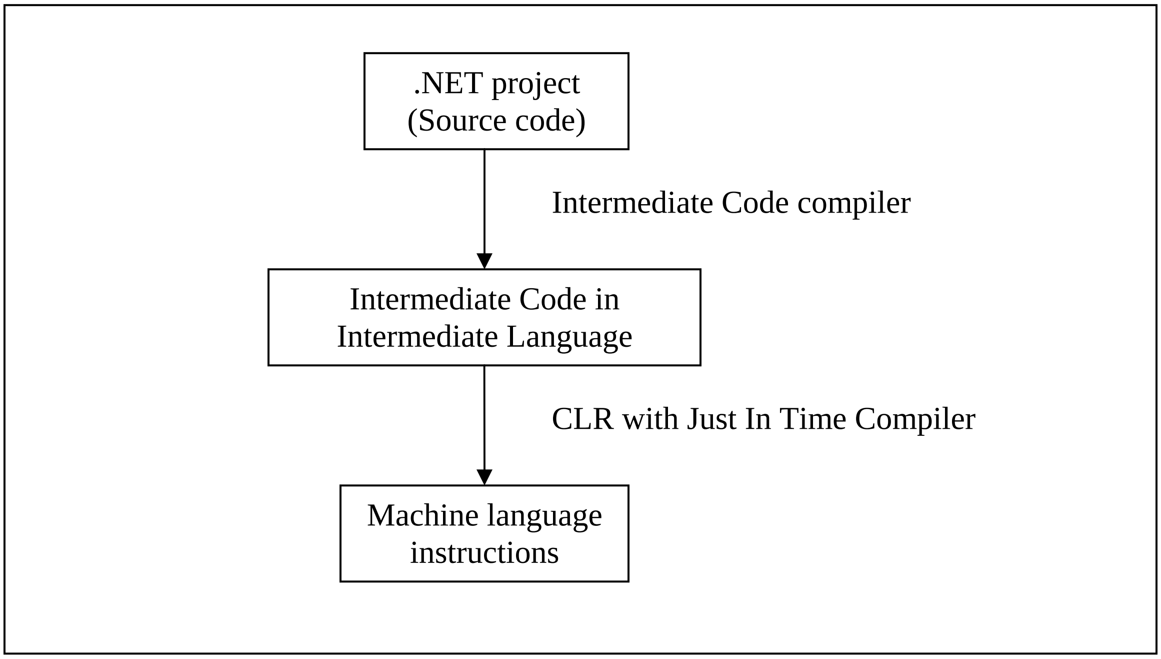


Figure 1: Illustration of compilation and running of .NET project

In Figure 1, any .NET project developed in C#, Visual Basic and others are first compiled (or build) by the Intermediate Code Compiler. The output of this is an intermediate code specified by Intermediate Language (IL). The IL is independent of any language constructs. As a result this IL code can be run on any Windows OS which can understand the IL constructs. For this CLR with Just In Time Compiler (JITer) is used also called the .NET runtime engine. The CLR understands the IL and then converts the intermediate code into machine code using JITer and then runs the application. For building the project Microsoft’s Visual Studio Integrated Development Environment (IDE) can be used. This will generate the intermediate code and produces the executable file in IL. So to run any .NET applications the user has to install the .NET runtime. Thus, Operating System and hardware system independence is achieved. OS independence and language interoperability are the main reasons for choosing to use this framework. Moreover, the WriteOn tool is designed to work as a plug-in with ConferenceXP client in the future which is developed in .NET Framework using C# language, Microsoft.Ink SDK and ConferenceXP SDK. C# language comes with lots of built-in packages when the .NET Framework is installed. These packages provide useful functionalities that can be helpful in building applications like WriteOn.

3.2 Technical terminology in the C# language

Some of the technical terms used in this thesis are defined as follows:

- *Class* – a class is a prototype of an object in an object-oriented language. A class may also be considered to be a set of objects which share a common structure and behavior. The structure of a class is determined by the class variables which represent the state of an object of that class and the behavior is given by a set of methods associated with the class.

- *Interface* – an interface is a prototype of a type of object to be created. It only declares the functions to be implemented by the *Class* which implements this interface. The difference between *Interface* and *Class* is that the methods are implemented in a class where as in an interface it is not.

- *Instance* – an instance, also called an object, of a class is a reference to the data of the class on which the program can operate by calling the functions defined.

- *Constructor* – a constructor of a class is used to create the instance of a class.

- *Namespace* – a namespace is the package hierarchy which contains classes and interfaces. Under a namespace there could be sub-namespaces also.

3.3 Configuring the .NET Framework and Visual Studio .NET projects

The .NET Framework must be configured to run WriteOn tool. The redistributable installation is available on [17]. Installing .NET Framework installs the runtime engine which runs the .NET executable applications and also loads the necessary class libraries. To configure a .NET project, first Microsoft Visual Studio (MS VS) .NET must be installed. MS VS product information, download information is available at

[18]. The installation file is available on the Microsoft website. This installation comes with Visual Studio (VS), an IDE for developing .NET applications. In VS there are several options to create new projects (single projects exist in a project file), new solutions (multiple projects exist in a solution), a new file for a project, etc. Once the project is created classes can be added to the project. Some of the class files require the .NET dynamic link library (dll) files to be used. This can be done by adding a reference to the projects by giving the location of the corresponding dll file(s). Once the class files are designed, they can be compiled to an execution files by selecting the Build Solution menu item on the main menu of VS. After compilation of the project is done VS generates an executable file that can be used to run the application. The application can also be debugged using the VS. The Microsoft VS .NET documentation provides further details.

3.4 ConferenceXP (CXP) and the ConferenceXP API

ConferenceXP is an initiative of Microsoft Research which helps in building distributed applications. ConferenceXP makes uses of tablet PC features and provides infrastructure to the ink-based applications. The SDK provides rich API to facilitate distance learning, collaboration and active conferencing in users' applications. CXP also makes uses of wireless technologies to provide applications build distributed systems. Since all the above mentioned capabilities are needed to build the WriteOn tool, CXP has become a natural choice. Material at [21] is the official website of ConferenceXP with latest information on client tools, news and developer community. Section 6.1.5 describes how CXP is used to build WriteOn.

Having looked at the .NET Framework and the terminology associated with writing object-oriented programs, let us now look at the major software components of the WriteOn system. Figure 2 contains a block diagram of the major components along with a schematic view of how the components interact. The remaining sections in this chapter describe the functionality of each of the major components.

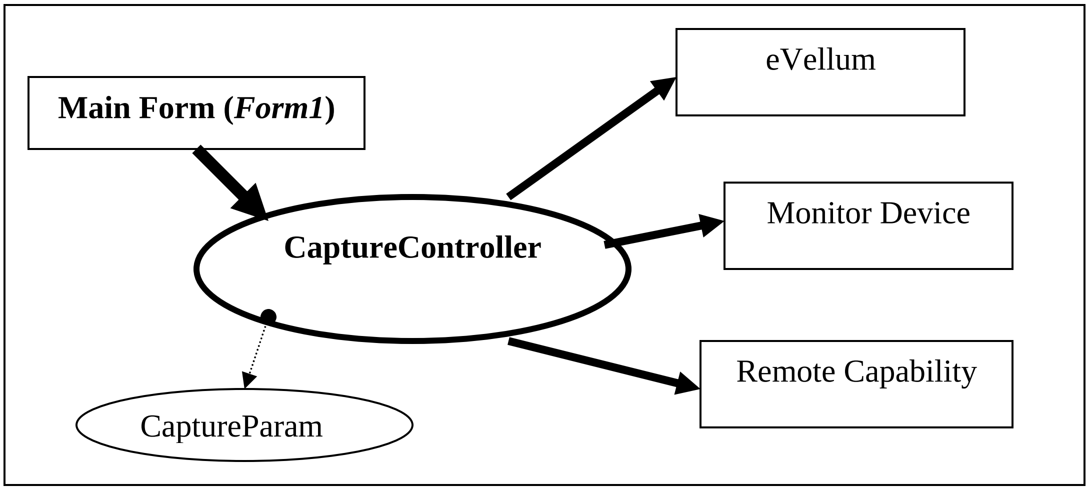


Figure 2: WriteOn system architecture

3.5 Structural Description of WriteOn

In Figure 2, the main form (*Form1* is the actual class) is the Graphical User Interface (GUI) application window of the WriteOn tool. When the WriteOn tool is started, the main form shows up. All the actions such as selecting properties for the tool, starting/stopping the capturing process, enabling/piercing the vellum and so forth are performed by selecting the various menu items and buttons from the main form. The WriteOn tool also sets an icon on the system tray (lower right portion of the system screen) to facilitate performing of several operational tasks within WriteOn.

Main form is just a User Interface (UI) component, which delegates most of the actions as functional calls to the *CaptureController* class, in Figure 2, to perform the required tasks. The *CaptureController* is a centralized controller that controls all other components of the software system. *Form1* holds a reference to the *CaptureController* in order to delegate events to the *CaptureController* in the future, created in the *Form1*'s constructor.

Various properties such as type of eVellum, audio/video options, whether to broadcast or store the video stream to a local filesystem and so forth are provided in the form of attribute values of the *CaptureParams* class instance. So if any of these properties are changed it is the object instance of the *CaptureParams* class which is changed. The delegates registered by the *CaptureController* notify other components of the WriteOn tool of the changes to these properties.

From Figure 2 one can say that the main form and other UI components (eVellum, Monitor Device and Remote Capability) form the “View”, *CaptureController* forms the “Controller” and *CaptureParams* forms the “Model” of the Model View Controller (MVC) architecture which is commonly used in building GUI-based applications. All the software classes required for developing these components are abstracted properly from each other so that only the required interfaces of a component are exposed to other components based on Object Oriented Paradigm (OOP). The OOP concepts are explained in C# programming language elaborately in [20].

Overall, this chapter discusses the basic structure of the WriteOn tool at the system level. Several features like achieving transparency, capturing the desktop screen activity, broadcast of the screen activity and others are handled by components like eVellum, monitor device, remote capability, and other components of the WriteOn tool. These above mentioned components are described in the next chapters.

Chapter 4

WriteOn Components

This chapter provides a brief introduction to the major logical components that provide the features available in the WriteOn tool. It also includes a brief discussion of the technologies used in building these components. A detailed description of each of these components is provided in Chapter 5.

WriteOn tool consists of three main logical components.

- Electronic Vellum
- Monitor Device and
- Remote Capability.

4.1 Major Components

4.1.1 Electronic Vellum (*eVellum*)

Electronic vellum is a virtual transparency film/surface/slide/layer that resides virtually on top of all applications windows currently running in the computer display screen. It allows the user to annotate on it using the electronic ink stylus. This means a user can see the applications running under the eVellum as well as the ink annotations on top of it. Chapter 5 gives a detailed explanation of the eVellum.

4.1.2 Monitor Device

The Monitor Device, a software module, allows the user to capture the desktop screen activity of a computer to a video file. It is also capable of broadcasting this desktop screen activity over the Internet port at the same time as the capture operation is taking place. Chapter 6 provides a detailed description of the implementation of the Monitor Device.

4.1.3 Remote Capability

The Remote Capability, a software package, is another important component of the WriteOn tool. It is the component most used by the students to view the ongoing instructor’s presentation. Technically it plays the video stream broadcast from the instructor’s end by the WriteOn tool. The internal implementation details for the Remote Capability module are elaborated on in Chapter 7.

4.2 Technologies Used

A Tablet PC provides an easy and flexible environment for presentation purposes. Tablets act as a basic notebook computer. However, its most important feature is that it can be folded to be in the tablet mode making it easy to use as a drawing/writing slate. Moreover, it is very light weight and therefore easy to carry and it has a lengthy battery life. The instructor can move around in the classroom carrying the tablet without much difficulty. A Tablet PC is provided with a stylus the user can employ to input electronic ink to an application capable of processing the stylus information. On a typical tablet screen there are several buttons that the instructor can configure to perform functions specific to each particular application, thereby making it easier to make operate the application when the computer is in tablet mode.

WriteOn is built using several Microsoft’s technologies. The important ones are: Microsoft.Ink SDK for Tablet PCs, DirectShow of DirectX, and Windows Media

Encoder (WMEncoder). DirectShow and WMEncoder are used in parallel to build the features of the WriteOn tool. The ConferenceXP API is also used to produce the broadcast of the video stream of the instructor’s desktop screen activity along with DirectX’s DirectShow package

4.2.1 Microsoft.Ink for Tablet PCs

Microsoft.Ink technology is used to achieve ink annotation on the eVellum. Microsoft.Ink is a Component Object Model (COM) technology used to aid in building ink-based applications. The package comes as a part of the Microsoft Tablet PC platform SDK. The package is designed to work with Tablet PCs, PDAs and other hand held computer devices which have digitizing screens and an electronic ink stylus as an input device. The package provides various classes and interfaces through which the applications can enable, receive and process the stylus events. Furthermore, the application can convert these raw stylus events into ink strokes thus allowing the user to annotate on an application window. The package also provides the default implementations of most of the interfaces such as *InkOverlay*, *InkCollector*, *Ink*, *Stroke*, *DynamicRenderer*, *RealTimeStylus* and others so that the developers can build ink-enabled applications with less effort.

4.2.2 DirectShow of DirectX SDK

Microsoft DirectShow version 9.0 SDK is also used to build the WriteOn software system. DirectShow is basically a filter-based framework, which facilitates generating a movie file of the desktop screen activity. DirectShow also performs media conversion (audio and video) of movie files. In conjunction with ConferenceXP APIs, DirectShow performs some of the work of broadcasting of the video frames to the student machines. Refer to section 6.1 about the DirectShow API and 6.1.1 through 6.1.5 about how the features of the WriteOn tool is achieved using DirectShow.

4.2.3 Windows Media Encoder

Windows Media Encoder is a technology based on Windows Media Format (WMF) Windows Media Format is an audio/video technology used to create applications that play, write, edit, encrypt, and deliver Advanced Systems Format (ASF) files and network streams. These media files and streams commonly contain audio and video content encoded using the Windows Media Audio and Video codecs. WMF is also used to achieve the same features as those of DirectShow of DirectX but using different file formats or codecs. Both DirectShow and WMEncoder each have several advantages and disadvantages, which are detailed in section 6.4. The main reason for using both of them is to compare the performance of each of these technologies to ascertain which performs better. Pros and Cons of the DirectShow and WMEncoder are described under the major components in the subsequent chapters.

After discussing briefly the major components of the system which provides features of WriteOn tool, Chapter 5 through Chapter 7 describe the technical details in implementing the eVellum, the Monitor Device and the Remote Capability.

Chapter 5

Electronic Vellum (*eVellum*)

WriteOn is based on the ability to place electronic ink anywhere on the computer screen. We call this ability Anywhere Annotation. To build the WriteOn feature of Anywhere Annotation (i.e., electronic ink annotations anywhere on the computer screen), this research has implemented an entity called *eVellum* on which a user can apply electronic ink. An eVellum is a transparent window, which allows the user to see the applications windows underneath it and also to annotate on the eVellum window. This can be visualized as a plastic transparency slide on the classroom projector used for presentations. This is a powerful feature for an instructor making a classroom presentation. Whenever the instructor opens a dynamic application (applications which animate or draw frequently such as LogicWorks simulator – an IDE used to design electronic circuits and simulate the circuit – MicroChip, Winamp, Windows Media Player) in order to introduce or elaborate on some subject matter, s/he can set the eVellum over these application windows and then start annotating on it.

eVellum is a transparent window or simply can be thought of as a separate application window. While annotating on the eVellum the instructor generally does not want to switch to different application windows or deactivate the eVellum accidentally by pressing Alt+Tab or by clicking the application status bar icon. To avoid deactivation of the eVellum, it is set as the topmost window of all of the application windows which are currently opened so that the instructor can continuously annotate on it.

In order to annotate over any type of applications, two types of eVellum are designed for the WriteOn tool. These are:

- Solid Vellum
- Overlay Vellum

Before explaining these vellums, let us look at the way application windows work in the Windows Operating System (OS). This will help in understanding why the WriteOn tool creates these two types of eVellums.

5.1 Windows OS application window's behavior

In the Windows OS, two types of windows can be created in general. These are: a normal window and the second type is an overlay window. The way these two types of windows behave is described below. Each window created in the Windows OS consists of a background layer and several other overlay layers on top of it. Normal windows render mostly on the background layer when the OS triggers paint events. Each pixel is a color structure which consists of Red, Blue, Green and an Alpha-bending values (also called the alpha component) ranging from 0 to 255. An alpha-bending value of 0 means a pixel is completely opaque while an alpha-bending value of 255 indicates that the pixel is completely transparent. Alpha-bending causes the see through effect on windows i.e., for pixels with larger alpha-bending value, one can see through the pixel of the application window to whatever window underneath it. Usually, all pixels of the normal window do not contain the alpha component. Therefore a normal window is always opaque in nature i.e., one cannot see any other background application window through a normal window. Application windows like Acrobat, WordPad, Notepad, Web browser and so forth fall into this category where there is no alpha bending component in the pixels that make up these windows. But pixels on normal windows can be seen through overlay windows. An overlay window has an alpha-bending component for all of its pixels and paints mostly on the top layer of the window (as opposed to the background layer). Because of the inclusion of the alpha-bending component for every pixel of an overlay window, a user can see the application windows underneath an overlay window.

The value of the alpha component is set by the *Opacity* property of the window in Windows OS. This value is within the range from 0 to 100%. When the *opacity* is 100% the window is completely opaque, giving the effect of a normal window. When the *opacity* is 0%, the window is completely invisible. However, when the *opacity* property is in between 0 to 100% e.g., if the value is 25% then it looks like a translucent window. In practice, the email notification popup of the Microsoft Outlook, which appears in the bottom right when a new incoming email arrives, falls in this category. This Outlook popup is a translucent window through which one can see the underlying application windows.

Although some overlay windows, such as Windows Media Player and Winamp, do not have an alpha component, they paint on the top layer rather than in the background layer. Such windows cannot be seen through any other overlays, because of the way the Windows OS functions. However, these overlay windows but can be seen through the normal windows with some modifications in the way the normal windows behave. These special behavior windows are called Solid Vellum and will be discussed in the next section. Since the classes in the WMEncoder technology do not support capturing these overlay windows, it is necessary to use Solid Vellum in order to perform the desired functionality of recording the electronic ink applied over overlay windows. Therefore, it was imperative to explore and develop two different types of vellum, which behave exactly the same way as far as the user is concerned. The following sections elaborate on the details of the two types of eVellum: Solid Vellum and Overlay Vellum.

5.2 eVellum Types

5.2.1 Solid Vellum

Paint events are triggered whenever a window is activated/deactivated, maximized, minimized or iconified, etc. These paint events are what render a window and its contents on the screen. OS Paint events generally contain information about which area to paint, which window to paint, along with a handle to the instance of the graphics object. Solid Vellum is similar to a normal window, but in order to see an overlay window such as Winamp, this eVellum does not paint itself when the Windows OS triggers paint events. Since the solid vellum is not painting the application window, the user can see the background application window behind a solid vellum window. However, the dynamic behavior of the normal window cannot be seen through the Solid Vellum. Since Solid Vellum does not paint itself, it cannot capture the modified regions of the background normal windows to show their dynamic nature, which explains why only the static picture of these windows is seen. Therefore only the Winamp kind of applications can be seen through this vellum.

Microsoft.Ink SDK is used to achieve ink annotation on Solid Vellum. The *InkOverlay* class of the *Microsoft.Ink* SDK is the main class which handles annotation. The instance of the *InkOverlay* class is created in the constructor of the Solid Vellum form by giving a handle to the Solid Vellum. In the Windows OS, a handle of a window is a reference that uniquely identifies a particular application window. *InkOverlay* then sets the pen properties including such things as color and width of the pen as specified by the user, after which the *InkOverlay* instance is enabled. Now the Solid Vellum is ready to receive the annotations of the real-time stylus.

5.2.2 Overlay Vellum

The behavior of the Overlay Vellum is little bit different than that of the Solid Vellum. It consists of basically two windows lying one over the other. A parent window lies behind a child window as shown in Figure 3. There is a reason behind using two windows to achieve transparency and is described below. Imagine if there is only parent window i.e., without the child window over the parent window. The parent window indicated as dotted rectangle in Figure 3 has the transparency key set as the background color. This causes the pixel to be essentially transparent. As a result, wherever there is a pixel of background color (equal to the *TrasnparencyKey* color) on the parent window the user can see through it. If the entire set of pixels making up a parent window is set this way, the parent window is entirely transparent. However, the disadvantage of this scheme is that the user cannot annotate on the parent window – it wouldn’t be seen since the Transparency Key is set to make the parent window transparent. Unfortunately, the Windows OS allows mouse or the stylus events to logically pass through the regions wherever the background pixel color is same as the *TransparentKey* color, thus making the underlying applications receive the events. Therefore, a parent window cannot be used by itself.

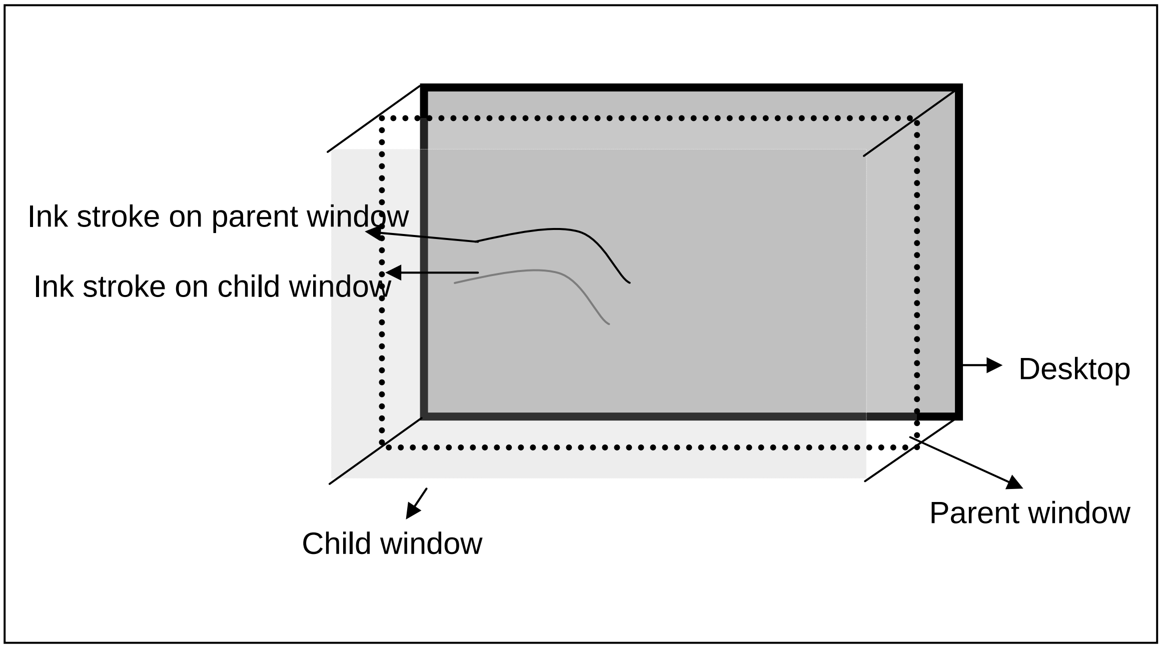


Figure 3: Illustrating child window overlapping the parent window over Desktop

For this reason a child window, shown in Figure 3 as the translucent window, is created by the *CaptureController* which resides on top of the parent window. The child window has the *Opacity* property set to 0.05%. This makes the child window close to transparent. However, in this situation even the ink becomes transparent because of the way in which the Windows OS Graphic Display Interface (GDI) operates. Figure 3 shows an ink stroke on the child window. This ink stroke is also transparent opening a view down to the parent window in the locations of the pixels that make up the stroke. Therefore, to achieve the transparent effect and yet have the ink annotations visible, non-transparent ink strokes are added on the parent window after they are received on the child window. This produces the dark ink strokes on the Parent Window of Figure 3, which appear as opaque and solid strokes. Because the stroke appears on both these overlapping windows at the same location the ink strokes do not appear transparent or distorted. The overall effect is the appearance that the ink stroke is placed on the application window, when in fact it is received on the child window (which is transparent), rendered on the parent window (which is also transparent), and appears over the top of the underlying application window. For ink annotations, the Microsoft.StylusInput Application Programming Interface (API) of the Tablet PC SDK is used.

5.2.2.1 Operation of the *RealTimeStylus* API

Information is input to the OS from the ink stylus through the operation of the RealTimeStylus API.

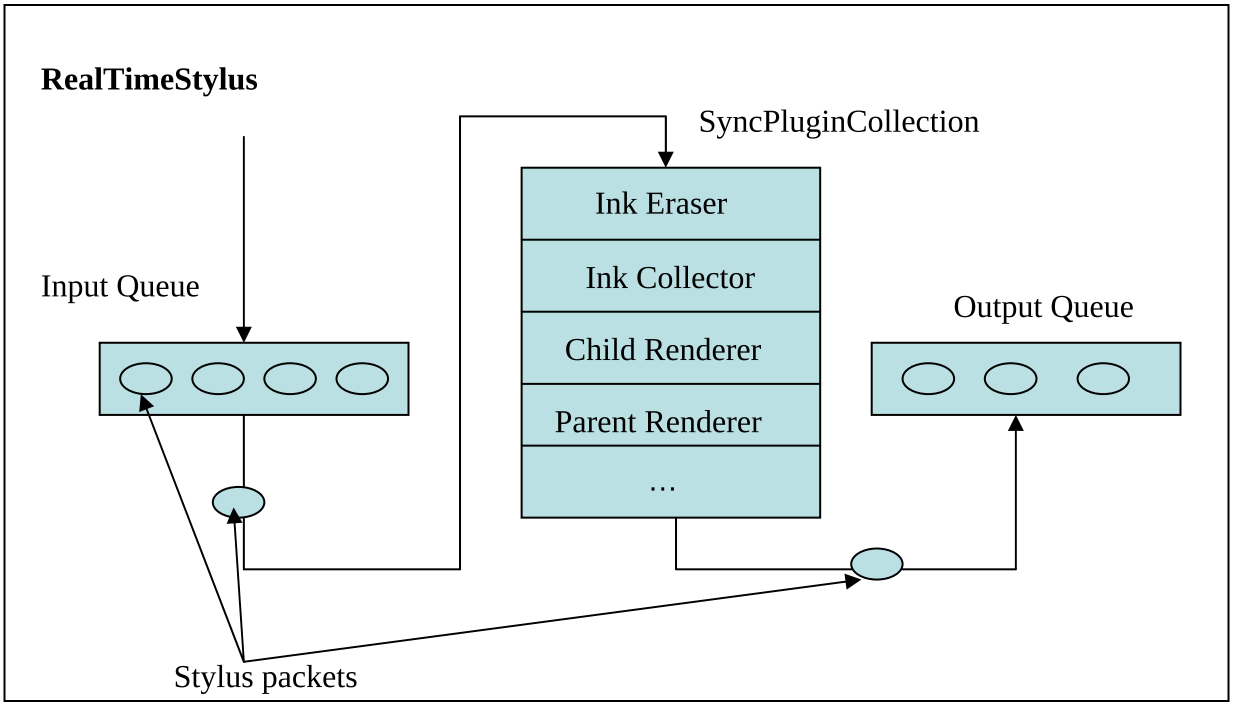


Figure 4: Work flow illustration of the RealTimeStylus

As shown in Figure 4 *RealTimeStylus* class of the *Microsoft.StylusInput* API has two queues: an input queue and an output queue. These queues contain the stroke packet information, which consists of the information about the location (X and Y position in the ink space coordinate system) of the real-time cursor, pressure information, whether the stylus is in inverted position, and so forth. Moving the stylus close to the digitizer or the Tablet PC monitor or touching the digitizer with it generates the stroke packets and pushes them into the input queue. At the same time as packets are entering the queue, the packets are processed by the instances of the *SyncPluginCollection* class as shown in Figure 4. The *SyncPluginCollection* class holds the instances of the *IStylusSyncPlugin* interface that actually processes the stroke packets. Basically, the pushing of the stroke packets into the input queue and the passing of the packets to the instances of the *IStylusSyncPlugin* class occurs in the same UI thread. For this reason they are called the synchronous plug-in collection. These plug-ins (*InkEraser*, *InkObject*, *ChildRenderer* and *ParentRenderer* in Figure 4) can modify packets or add new packets to the input queue or delete packets from the input queue and finally send packets to the output queue. There are other *AsynchPluginCollection* objects that process the packets in the output queue later which are not of concern for this work. The next section explains how the above technology achieves ink annotation.

5.2.2.2 Ink annotation using *RealTimeStylus* API

When the user starts inking on the surface of the digitizer, the *InkEraser, as in Figure 4*, is disabled because the user is inking (Ink Mode), and the *InkObject* starts collecting the packets and also passes the packets to the renderers. Both the *ChildRenderer* and the *ParentRenderer* draw the ink strokes on the child window and the parent window based on the stroke packet information, respectively. When the user lifts the stylus the packets collected to that point by the *InkObject* are stored as a stroke in the instance of the *Microsoft.Ink* class which both the child and the parent windows share. Storing of the ink strokes at the instance of the *Microsoft.Ink* class allows the system to re-render the strokes when the window repaints or refreshes. When the user starts erasing the strokes (Erase Mode), the *InkObject* is disabled, and the *InkEraser* checks the ink strokes which are near the electronic stylus (now an eraser) and deletes them. In turn, the *InkEraser* deletes the stroke packets since the child and parent renderers should not paint the erased strokes. All ink annotations on eVellum are produced by the above procedure.

5.3 eVellum Modes

eVellum can be operated in two modes:

- Vellum Mode
- Pierce Mode

In Vellum Mode, the eVellum is enabled and it resides on top of all running application windows. In this mode, the user can annotate and erase the ink on the vellum. In the Pierce Mode, the vellum is disabled and any ink that has been previously applied to an eVellum is removed from the screen. The Pierce Mode is one of the important features of the WriteOn tool because in this mode the instructor can run and operate/manipulate any other application relevant to the lecture subject for explanation purposes, and then switch back to the Vellum Mode for annotation. The above mechanism therefore, adds more flexibility and usability to the active lecturing process than a static PowerPoint presentation can provide.

Toggling between the Pierce Mode and the Vellum Mode is done by selecting the Pierce Vellum menu item of the WriteOn tool menu found in the System Tray. The Pierce Mode can also be set by pressing a hot key (right now it is <F2> key) configured as part of the initial startup of the WriteOn tool. From the software point of view, switching between modes is done by hooking to the Windows system wide events. Windows OS supports an entity called *Hooks*. Whenever any event occurs in the OS (like mouse events, keyboard events, paint events, Windows creating and deletions events, and so forth) the events are pushed into the event queue. An application can receive only certain specific kinds of events (like keyboard events) by registering a connection to them using hooks. This is done by calling *SetWindowsHooksEx()* of the Windows API and providing the program function handle to receive the events. The WriteOn tool registers to the keyboard hooks in order to listen on a specific registry key. Whenever the F2 hot key is pressed, the application receives the event and switches between the Pierce and the Vellum modes. The toggling between the Pierce and the Vellum modes by hooking into the Windows OS keyboard events is an important aspect used to improve WriteOn usability. In order to fully use this usability feature when operating in Tablet Mode, the hot key can be mapped to one of the special purpose screen buttons that appear on the display of the typical Tablet PC. However, these special buttons are different for each Tablet PC hardware manufacturer and the mapping connection between the F2 keystroke and the special purpose button must therefore be done by the user before operating WriteOn. Using this feature, the instructor, does not need to click the application menu item in the system tray bar at the bottom of the screen every time to switch between modes. Instead, the instructor can simply press the special button on the Tablet PC screen (slate) Appendix I describes the functions involved in achieving this feature.

5.4 Snapshot Feature

When the instructor wants to preserve the ink strokes and the background information for future lectures, s/he uses the snapshot feature, which allows him or her to take a snapshot of the desktop. Snapshots are stored as image files. The first time a user takes a snapshot during a WriteOn session, the system prompts the user for a directory in which the images are to be stored... The user can also set the snapshot directory in the properties dialog box of the WriteOn tool. See section 8.1.1 for further details as to how to take snapshots. The feature is implemented by making calls to the Windows OS from the C# code using native method calls. Appendix II describes the programming details involved in achieving this.

Apart from the above features, the instructor has the facility to change the pen color and width used for ink annotation for lecturing purposes. Section 8.1.1 describes about how to set these properties.

Chapter 6

Monitor Device

The Monitor Device package of the WriteOn tool solves the problem of capturing and saving an instructor's presentation. The capturing and saving of an instructor's presentation is important for effective teaching and learning. Using a recorded movie of the annotation process means an instructor does not need to prepare the presentation every time for the same lecture; instead he/she can distribute the lecture video file to the students who can then view the presentation. This saves preparation time for the instructor. Movies of screen activity also allow students to review the complete record of classroom lecture screen activities at their own pace and at a time of their choosing.

Most of the time instructors or conference presenters use different sets of computer tools, for example use of PowerPoint for presentation and Camtasia® Studio for capturing, to present the subject matter and to save the presentations to a media file. The WriteOn tool gives an advantage over other tools in this regard, in that the instructor need not put time and effort in installing and learning a variety of different computer tools.

In response to the above considerations, the "Monitor Device" component of the WriteOn tool is designed to be a capturing mechanism. The Monitor Device currently provides the following features:

- Capturing and saving the desktop screen activity and the instructor's annotations to an Audio-Video Interleaved (AVI) file.
- Saving the instructor's voice from the audio device attached to the instructor's Tablet PC or NoteBook along with the desktop screen activity.

- Capturing and saving any other application window to a movie file (even though the user interface is not exposed to the user currently).

- Broadcasting the samples of the captured desktop screen activity while capturing, so that students can view what is happening at the instructor's Tablet desktop in real-time in the classroom.

After reviewing several products used for classroom presentation (see [5], [6] which provide links describing the products and other information) and technologies, two technologies: Microsoft's DirectShow and Windows Media Encoder were used to build the Monitor Device. Although both these technologies are used to provide the same set of features for the initial version of the WriteOn tool, only one of them, the technology which gives better performance and supports more features, should be used for future development of the WriteOn tool.

The following sections review the various implementation details of the Monitor Device, first using DirectShow and then using Windows Media Encoder in order to highlight the advantages and disadvantages of each of these technologies. In this way, it is possible to identify which of these two technologies is the best choice for future enhancement of the WriteOn tool.

6.1 DirectX's DirectShow

Since DirectShow is one of the two technologies used to build the Monitor Device, it is important to understand how this technology, developed by Microsoft, works. The Microsoft® DirectShow® application programming interface (API) is a media-streaming architecture for the Microsoft Windows® platform. Using DirectShow, applications can perform high-quality video and audio playback or capture. 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. It supports capture using Windows Driver Model (WDM) devices or older Video for Windows devices. Integrated with other DirectX technologies, DirectShow automatically detects and uses video and audio acceleration hardware when available, but also supports systems without acceleration hardware.

DirectShow simplifies media playback, format conversion, and capture tasks. At the same time, it provides access to the underlying stream control architecture for applications that require custom solutions. A user can create DirectShow components to support new formats or custom effects. All of these features made DirectShow seem like the most advantages choice for building the Monitor Device.

6.1.1 DirectShow Filters

DirectShow is based on the Component Object Model (COM). The basic software component model used in DirectShow is called a *filter*. A filter is a software component that performs some operation on a multimedia stream. For example, DirectShow filters can

- read files
- get video from a video capture device
- decode various stream formats, such as MPEG-1 video
- pass data to the graphics or sound card

Filters receive input and produce output. For example, if a filter decodes MPEG-1 video, the input is the MPEG-1-encoded stream and the output is a series of uncompressed video frames.

In DirectShow, an application performs any task by connecting chains of filters together so that the output from one filter becomes the input for another. A set of connected filters is called a *filter graph*. For example, the Figure 5 shows a filter graph for capturing the output from the webcam to an AVI file.

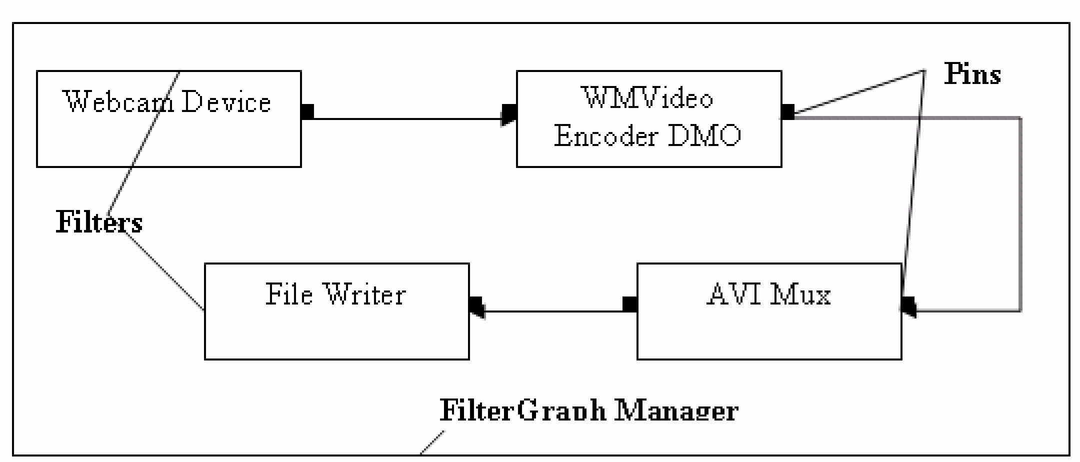


Figure 5: Filtergraph used to capture the output of a webcam

The WebCam Device filter in Figure 5 pushes the video frames captured from a webcam to the WMVideo Encoder DMO filter. The encoder filter is a compression filter which compresses the arriving samples and then pushes them to the “AVI Mux” filter in Figure 5. AVI Mux filter then formats the incoming compressed video samples into an AVI media format and passes these modified samples to the next filter. The “File Writer” in Figure 5 filter then writes the incoming AVI samples to the specified file. Filters are connected by pins, which are 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 to which they are connected. The input and the output pin must agree on a common media format in order to connect to each other. All the filters are contained in the FilterGraph Manager component.

The application does not have to manage all of this data flow. Instead, a high-level component called the Filter Graph Manager controls the filters. The application makes high-level API calls such as "Run" (to move data through the graph) or "Stop" (to stop the flow of data). For more control over the stream operations, the user can access the filters directly through Component Object Model (COM) interfaces. The Filter Graph Manager serves another purpose as well: It provides methods for the application to build the filter graph, by connecting the filters together.

Since DirectShow is based on COM, all the objects and resources created while creating the graph have to be released after use since they consume a significant amount of OS resources.

For use in WriteOn to capture the desktop activity, the user has to create a filter graph manager with appropriate filters and then run the filter graph manager. The filter graph used for creating the Monitor Device in the following section

6.1.2 Filter graph of the Monitor Device

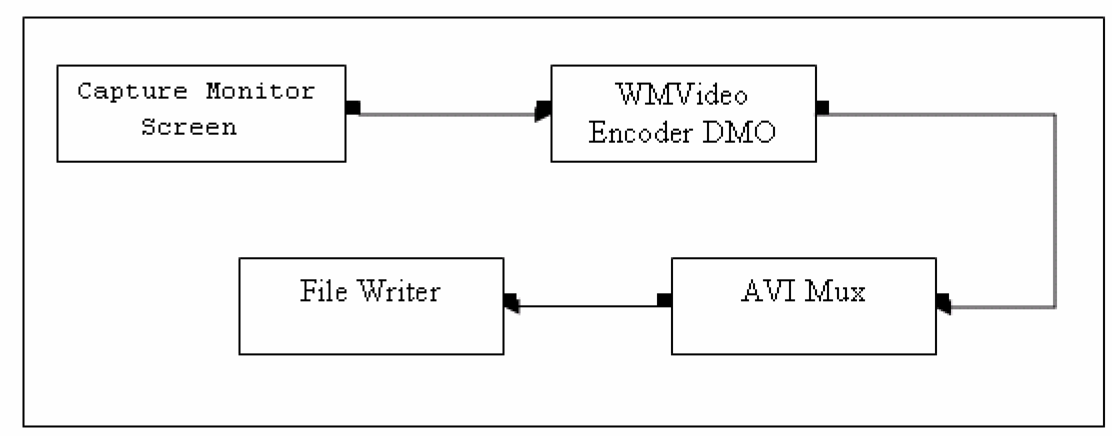


Figure 6: Filter graph used to capture desktop screen activity

Figure 6 shows the filter graph used for capturing and saving the desktop screen activity. The “Capture Monitor Screen” filter in Figure 6 takes the screen shots of the desktop and passes the sample bit information to the WMVideo Encoder DMO, which compresses the samples. The rest of the filters are the same as those in Figure 4. Several video compression filters, such as WMVideo Encoder DMO, WMVideo9 Encoder DMO, Indeo video 5.10, MSScreen encoder DMO, and so forth were tested, but WMVideo Encoder DMO gave better results in terms of performance and quality. Monitor Device, therefore, uses WMVideo Encoder DMO as the default compressor.

In order to achieve better quality of the movie file the instructor has to set several properties of the compressor. This is done by entering values in the properties dialog box of the WriteOn tool under “Video” tab menu. See Section 8.1.2.1 and Figure 16 about how to set these properties.

Properties of the compressor filter:

- Compression Width is the width in pixels of the compressed samples arriving at the input filter.

- Compression Height is the height of the compressed samples arriving at the input filter.

- BitRate is the number of bits per nanosecond that the output pin of the compression filter emits.

- Frames/Sec is the number of frames per second that the output pin of the compression filter sends to the next filter.

- Compressor option allows the user to enter any custom or third party DirectShow video compressor filter for better quality of the generated video.

The following section describes the working of the Capture Monitor Screen filter.

6.1.3 Capture Monitor Screen Filter

Capture Monitor Screen Filter (Capture Filter) is a modified version of one of the filters provided with the DirectShow filters. It is modified to achieve capturing of the desktop screen or any application window. The filter graph gives a callback to Capture Filter to fill the video frame information such as width, height of the video frame, color of each pixel, the video frame number, etc at every interval as specified the bit rate value. The Capture Filter then takes the desktop screen shot using Graphical Design Interface (GDI) operations and fills the video frame buffer with pixel information including the width and height of the screen shot, the type of media format such as 24 or 32-bit color depth, and so forth. The media sample is then sent by the Capture Filter across to the compressor for image compression. For the Capture Filter to work properly, properties such as the desktop window handle or any other application window handle (used for capturing a specific window), the dimension of the compressed video frame and the frame rate is set by *CaptureController*. Since Capture Filter is implemented in C++ and the rest of the WriteOn tool is developed in C# language, Capture Filter operates through a COM interface. By doing so, the .NET runtime allows for interoperability between C# and C++ through a COM wrapper.

To create the COM interface, an Interface Definition Language (IDL) file was written and then processed by the .NET IDL compiler to generate the interfaces and the Dynamic Link Library (dll). The capture filter implements the COM interface thus generated and the Monitor Devices uses the dll to make the function calls on the capture filter using the interface exposed as a COM interface. The steps involved in the development of the idl file, interfaces and the COM dll are covered in Appendix III.

6.1.4 Capturing desktop screen activity and the voice of the instructor

To capture the voice of the instructor, the instructor’s Tablet needs to have an audio device, a microphone, attached to it. The *Audio* tab of the properties dialog box of the WriteOn tool displays the number of audio devices attached to the Tablet. In it, the instructor has to enable the audio option and select an audio device in order to record his/her voice.

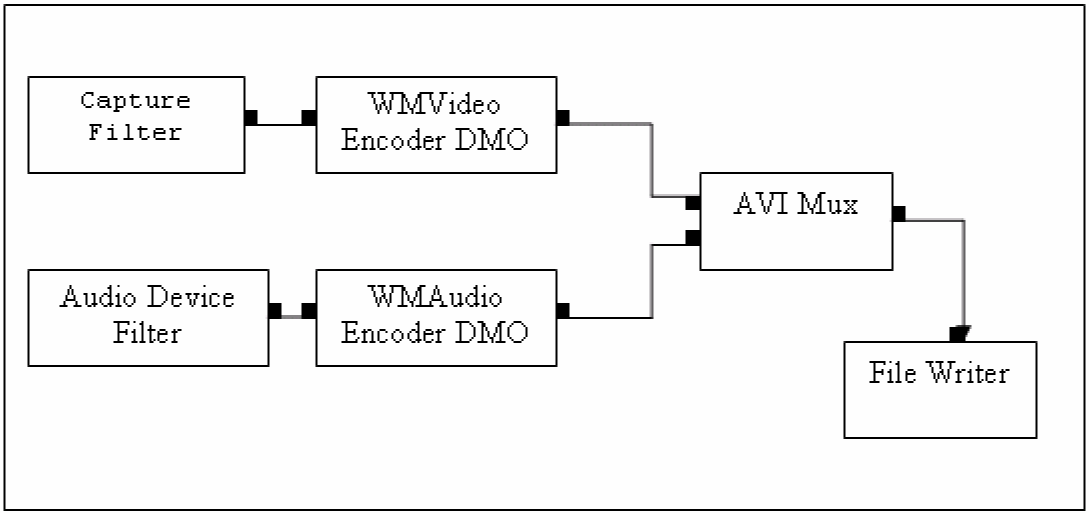


Figure 7: Filter graph to capture desktop screen activity and audio

The filter graph used for capturing the presenter's voice is more complex than the simple desktop screen capture and is shown in Figure 7. The filters used to capture the desktop screen activity remain the same and an Audio Device filter is added to the filter graph. The Audio Device filter in Figure 7 is connected to the audio compression WMAudio Encoder DMO filter to achieve compression of the audio frames. Both the audio and video compression filters are connected to the AVI Mux filter to produce AVI samples. On *Run()* function call on the filter graph instance by the *CaptureController* starts the capturing of the desktop screen activity and also records the voice of the instructor.

6.1.5 Broadcast of the instructor's presentation

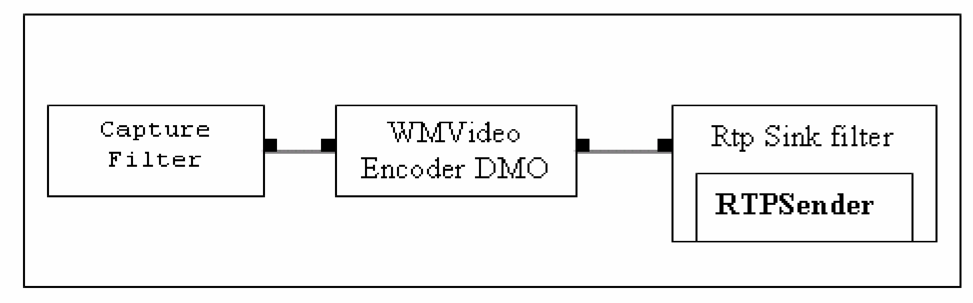


Figure 8: Filter graph used for broadcast of the instructor's desktop screen activity

When the user requests that WriteOn broadcast video samples of the real-time screen activity of the instructor, the AVI Mux and the file writer filters in the filter graph of Figure. 6 are replaced with *RtpSink* filter of the ConferenceXP as shown in Figure 8. The *RtpSink* filter basically serializes the samples and sends them across the multicast session group using *RTPSender*. The following section describes briefly how ConferenceXP API achieves broadcasting.

6.1.5.1 Overview of the ConferenceXP API

For broadcast of the video samples to take place the instructor has to connect to a "ConferenceXP Venue" (or simply Venue). A venue is a multicast group address used by ConferenceXP to connect several machines together. The WriteOn tool provides two static venues, one of IP address 234.4.4.4 and one of IP address 234.5.5.4. These options are available as menu items of the "Connect" main menu. Once a venue is selected, the *CaptureController* connects the tool to that venue using the name of the instructor or the student provided in the properties dialog box. The connection is created by creating an instance of the *RTPSession* class giving the friendly name (a user defined string used to signify that the RTPSession is created by based on that string) of this WriteOn tool.

CaptureController holds a reference to this *RtpSession* object until the user disconnects from the venue. From the *RTPSession* object, an instance of the *RTPSender* class is created and used for streaming the video samples that are arriving at the input to the *RTPFilter*. Therefore, the *RTPFilter* instance is created and is initialized with the *RTPSender* for the filter to push the samples into the network.

Using this relatively simple scheme based on the ConferenceXP API classes,, broadcasting of the instructor's desktop screen activity is implemented. The next section describes how Windows Media Encoder is used to provide features of the WriteOn tool.

6.2 Using Windows Media Encoder (WMEncoder)

The WMEncoder SDK enables the multimedia application developers to integrate Windows Media into their applications and workflow. Many different applications can be developed using the WMEncoder, such as screen capture utilities, digital recorders, custom encoding and broadcasting. The WMEncoder SDK is built on the Windows Media Format SDK, which provides the lower-level functionality of the various media formats. The WMEncoder has to be installed in order to use the WriteOn tool.

The Monitor Device, in the case of a Windows Media Encoder, is an *encoder* object. An *Encoder* is a software programmable component containing several *Source Groups.* A *Source Group* is a simple collection of information from several media sources which the encoder object encodes. Media sources can be either an audio source or a video source or any other source, including the desktop screen source. A media source is a reference to the types of sources attached to the machine, such as webcam, desktop, microphone, speakers, and so forth. Figure 9 provides an overview of the *Encoder* class.

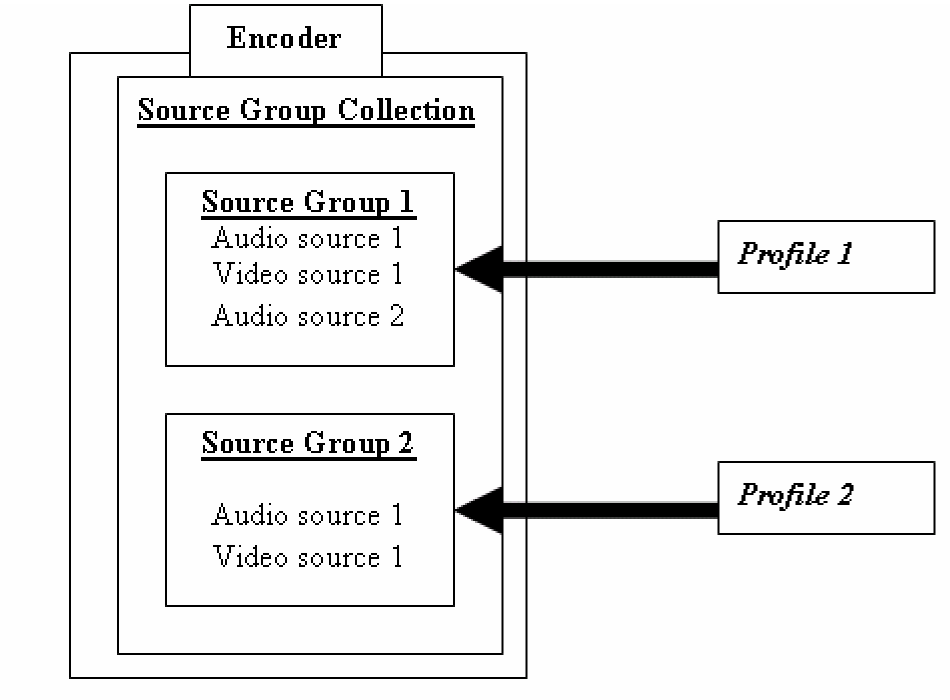


Figure 9: Windows Encoder object components

As implied pictorially in Figure 9, for the *Encoder* to start encoding, each source group must have a *profile* set. The Monitor Device sets the profile on the instance of the Encoder class. A *Profile* (shown in Figure 9, Profile 1 and Profile 2 are two different profiles supplied to the Encoder object) can be viewed as the media properties. These include specifications such as width and height of the samples, frames rate, the audio or video codec used for encoding, quality and sharpness of the movie generated, and so forth. Once the *profile* is set, the *Encoder* is ready to start encoding the desktop screen activity and to create a movie file. *Encoder* function calls such as *start()* and *stop()* govern the capturing process.

6.2.1 Capturing desktop screen activity and presentation audio

For capturing the desktop's screen activity, the desktop source is used as the video source with the "***Windows Media Video 9 Screen***" codec. To include the voice of the instructor, the audio device source is added to "***Windows Media Audio 9***" codec along with the desktop source. The file path of the movie to be generated is set on the *Encoder* instance before its starts encoding.

6.2.2 Broadcast using WMEncoder

Broadcasting the screen activity using WMEncoder is a very simple task. To do this the *Encoder* in Figure 8 has to be set with the URL (IP address and the port). This sets the *Encoder* to listen on that URL and for every external connection to the URL, the *Encoder* will send video samples to the client who requested the connection. This is a great advantage as any browser can play such a stream since most of them are equipped with media player plug-ins. Another advantage is that if the URL is available as public port over the Internet anyone who is remotely connected to this particular URL can view the ongoing presentation. However, this process does consume large amounts of bandwidth since the network transactions are done in unicast.

Although, there are several positive aspects to using WMEncoder and DirectShow technologies, capturing of the desktop background becomes very slow in many cases. Some of the major factors causing slow capturing are:

- Highly complex animations on the desktop
- Running gaming applications
- Extensive time spent on compressing the video/audio samples because of the nature of the video image
- Hardware acceleration for the display device set to a high level (forcing the make extensive use of the system hardware to accomplish video tasks)

Although it is possible to avoid the first 3 problems one way or the other, the last factor affects the capturing processing the most. The next section explains why and how hardware acceleration on the video card affects the capturing process.

6.3 Hardware (or) Video Acceleration (HA)

Most Tablet PCs, Notebooks, Desktop PCs, and so forth come with a video card. These video cards handle most of the graphics rendering operations. During a particular rendering operation, such as drawing a line, movement of an animated object, and so forth, the video card locks the parts of the screen region to create smooth and continuous rendering of a graphic operation. Because of these locks, capturing processes eventually have to wait until the entire region is unlocked. Obviously this causes a major delay during capturing. However, the HA in Windows OS can be set to 6 different values ranging from 0 to 5, 0 being the highest and 5 being the lowest. With a low value set for HA, the rendering operations are primarily handled by the processor (CPU) which does not cause many locks, thereby increasing the capturing process activity and speed.

Whenever the user starts the capturing process, the *CaptureController* automatically sets the HA of the primary display device, which is the monitor, to 5th level

(HA is full when the value is 0 and it is none when the value is 0). The user can set the HA level to the desired value in the properties dialog box. Programmatically it is done by setting the calling Windows API using interoperable services of the .NET framework and setting the registry key "Acceleration.Level" value of the primary device to level 5 (by default). Once the registry value is set, the Windows OS is notified about the change of the HA so that the OS can perform the rendering operations by the CPU from now on. The HA is reset to the previous value on stopping the capturing process. Appendix IV provides further details on Hardware Acceleration. Note that if there is an abnormal termination of the WriteOn software, the user must restore the appropriate value for HA by hand in the OS display properties dialog window.

6.4 DirectShow and WMEncoder Pros and Cons

DirectShow is a very flexible API which allows applications to integrate with lot more devices than WMEncoder. Using DirectShow, applications can capture the overlays also but one cannot with the WMEncoder. Since most application windows are visible through the Overlay Vellum, WMEncoder becomes a major disadvantage in this regard. But WMEncoder gives a better quality of the movie generated while capturing the desktop screen activity. In general if Solid Vellum is used for ink annotations its better to use WMEncoder but if Overlay Vellum is used DirectShow must be used.

Till now we have covered how the functionalities of the Monitor Device are achieved, described about Hardware Acceleration and how to disable it, described how DirectShow and WMEncoder technologies are used to achieve the WriteOn features, their pros and cons. In the next chapter we discuss about the Remote Capability, the last major component of the WriteOn tool.

Chapter 7

Remote Capability

Remote Capability is the last major component of the WriteOn tool. The main function of this component is to play the video stream, broadcasted at the other end by the WriteOn tool. Remote capability facilitates the students' connection to the instructor's presentation in the classroom. With this connection, students can view the presentation at the same time and so they will be in the same context as that of the instructor. Moreover, students can take notes by annotating on the window showing the instructor's presentation. Since students can take notes right away on the presentation, they can save time by not typing in Notepad or a WordPad and also do not miss the lecture.

The Remote Capability component also allows the students to save the whole lecture with instructor's annotations and their notes as a movie file in their local file system. They can later review the presentation by simply playing the saved movie file. Obviously this helps in students understand the subject more easily. Storage of their notes in the movie file and not in other documents provides an additional convenience to the students.

Like the Monitor Device, Remote Capability uses DirectShow for playing the video stream broadcast by the WriteOn tool using DirectShow technology. And Windows Media Player is used to play the video stream broadcast by the WriteOn tool using Windows Media Encoder. The following sections cover each of the above mentioned technology implementations.

7.1 DirectX's DirectShow

The RemoteDesktopController class controls the creation of the Remote Capability component. Whenever a student connects to the venue to which the instructor's is connected, an instance of the *RTPSession* class is created by the *CaptureController*. The *CaptureController* gets the notification about the instructors who are connected to the same venue by the ConferenceXP notification system. The *CaptureController* then checks whether the WriteOn tool is in instructor mode or student mode and gives the control to the *RemoteDesktopController* if the WriteOn tool is in student mode (if the WriteOn tool is capturing then it is in instructor mode otherwise it is in student mode). The *RemoteDesktopController* then creates an instance of the Remote Capability component for the stream (*RTPStream* class of the ConferenceXP), which the instructor is broadcasting. If there are multiple instructors connected to the same venue, then students on that venue view all of these presentations.

Remote Capability has two main classes. One is the *RemoteCapability* class, which builds the DirectShow filter graph to start, stop and pause playing the video stream. The other is the *RemoteDesktop* class, a window form on which the video of the instructor's presentation is played.

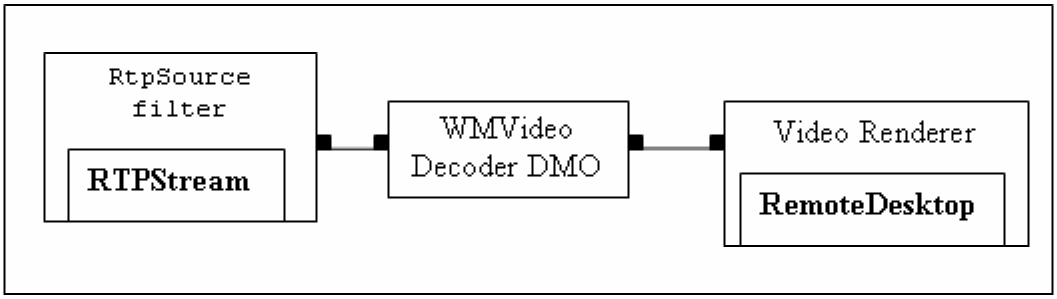


Figure 10: RemoteCapability filter graph

Figure 10 shows the filter graph used by the *RemoteCapability*. The *RtpSource* filter (a ConferenceXP RTP filter) in Figure 10 of the Remote Capability works exactly the opposite way from the *RTPSink* of the Monitor Device. It is initialized with the *RTPStream*, an input stream from which *RtpSource* filter reads the video samples. The

RemoteCapability builds the filter graph on receiving the first video sample, because it has to know the media format details such as width and height of the video image, bitrate, frames/sec and other information. The *RtpSource* filter in Figure 10 is connected to WMVideo Decoder DMO which decompresses the compressed samples to the video samples and then pushes them to the Video Renderer filter. Video Renderer plays the video samples as a movie. It is initialized with the instance of the *RemoteDesktop* class which is a window form on which video is rendered. Thus, playing of the broadcast instructor's presentation is achieved.

7.1.1 Annotating on *RemoteDesktop*

Similar to the Overlay vellum, annotating on the *RemoteDesktop* shown in Figure 10 is achieved by double window mechanism. *RemoteDesktop* acts like the parent window and *RemoteDesktopOverlay* overlaps on top of the *RemoteDesktop*. The reason for using the *RemoteDesktopOverlay* is that the *RemoteDesktop* is used for playing the video broadcasted and due to this the ink strokes on it annotated by the students gets erased whenever a new frame is being rendered on it. The filter graph manager shown in Figure 9 for playing the instructor's presentation does not send any events to the application layer about the rendering operations by the video renderer. So Remote Capability has no way to know, which region of the *RemoteDesktop* is painted. Effectively this means the students' annotations on the *RemoteDesktop* are not visible for more than few seconds even though they are actually present. To overcome the above mentioned problem, *RemoteDesktopOverlay,* a child window which overlaps *RemoteDesktop,* is created with *TransparencyKey* color set to its background color. By setting TrasparencyKey property of the *RemoteDesktopOverlay* the window becomes transparent. So *RemoteDesktop* is appears through the *RemoteDesktopOverlay*. Whenever, the *RemoteDesktop* is moved, resized, minimized, activated or deactivated *RemoteDesktopOverlay* is also performed respectively thus giving one window appearance. Microsoft.InkOverlay is used to achieve ink annotations on the *RemoteDesktop* and *RemoteDesktopOverlay*.

Whenever, the student starts annotating on the *RemoteDesktop, InkCollectorNewPacketsEventArgs* event occurs giving the ink stroke packet information. The *RemoteDesktop* uses this information to draw the stroke again on the *RemoteDesktopOverlay*. To improve the performance, the stroke is not drawn entirely rather it draws only from the most recent point to the current point at which the stylus is touching the monitor digitizer. Both *RemoteDesktop* and *RemoteDesktopOverlay* share the *Microsoft.Ink* instance which stores the strokes. Effectively *RemoteDesktopOverlay* is used primarily to show the ink annotations on the *RemoteDesktop,* where the ink strokes becomes invisible due to the playing of the broadcasted video of the Instructor's presentation.

Monitor Device is used again by the Remote Capability to save the video to a movie file.

7.2 Windows Media Player (WMPlayer)

WMPlayer is used to play the video stream when the instructor uses WMEncoder technology to broadcast the presentation. With this option even a browser which has any media player plug-in installed can play the broadcast video stream. Playing is handled by the instance of the *WMPlayerForm* class. The *WMPlayerForm* is an application window that creates the ActiveX WMPlayer component. This component is added to the controls of the *WMPlayerForm*. The ActiveX WMPlayer component is initialized with the URL at which the video stream is broadcast. The WMPlayer then plays the video by making the *start()* function call on the ActiveX component. The ActiveX WMPlayer component provides the start, stop and pause functions to control the playing of the video.

There are two disadvantages to the WMPlayer. One is that it will not allow ink annotating on the *RemoteDesktop* due to the way the ActiveX WMPlayer component renders the video stream. Also, the student one cannot save the presentation.

Chapter 8

WriteOn Tool Usage

The following sections explain how the instructor and the students can use the WriteOn tool in a classroom.

8.1 Instructor usage of the WriteOn tool

The instructor has several ways to use the WriteOn tool for classroom presentations. In the simplest case, s/he can just use the vellum for simple screen annotation. To do this, after starting WriteOn, the instructor selects the *File => Show Vellum* option as in Figure 11, which will set the vellum on top of all other active applications. Once the vellum is set, the main WriteOn form automatically minimizes itself so that the background applications' windows are fully visible. The *Show Vellum* option is also available in the WriteOn tray bar icon as shown in Figure 12. The WriteOn icon in the tray is opened by right-clicking on it.

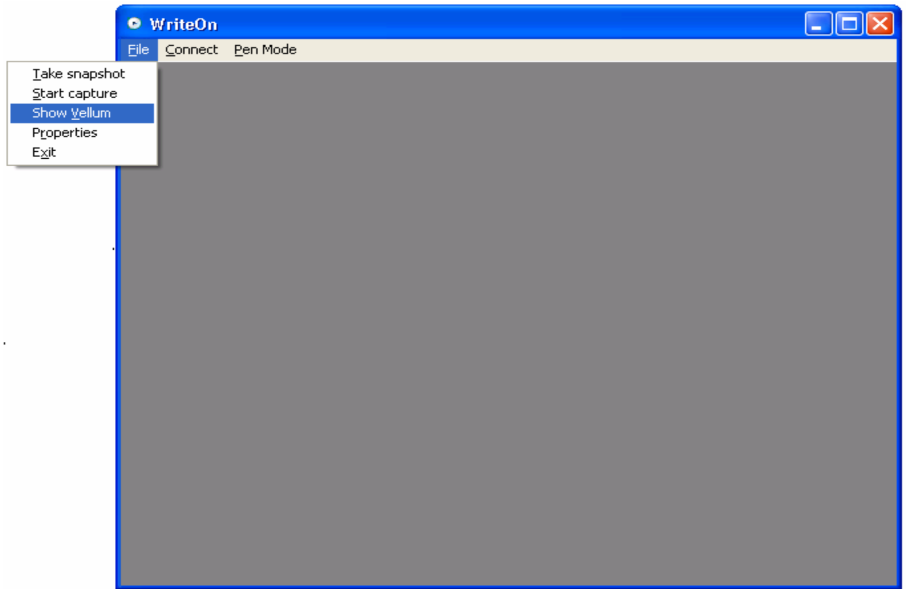


Figure 11: Show Vellum option on the File menu of WriteOn tool

Several other important options are also available in the tray bar menu of the WriteOn tool and will be explained in the following sections.

8.1.1 Setting the WriteOn properties

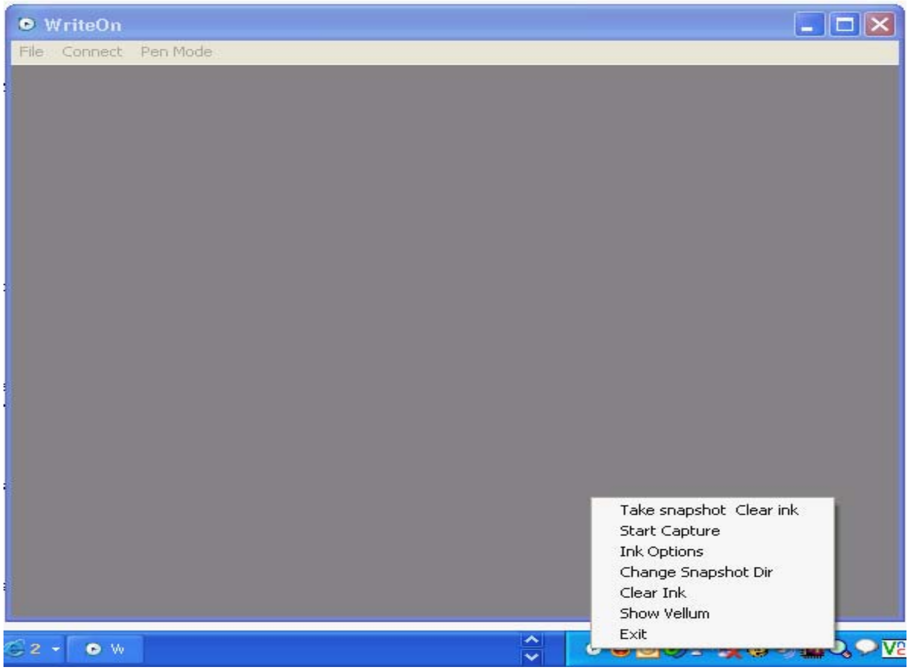
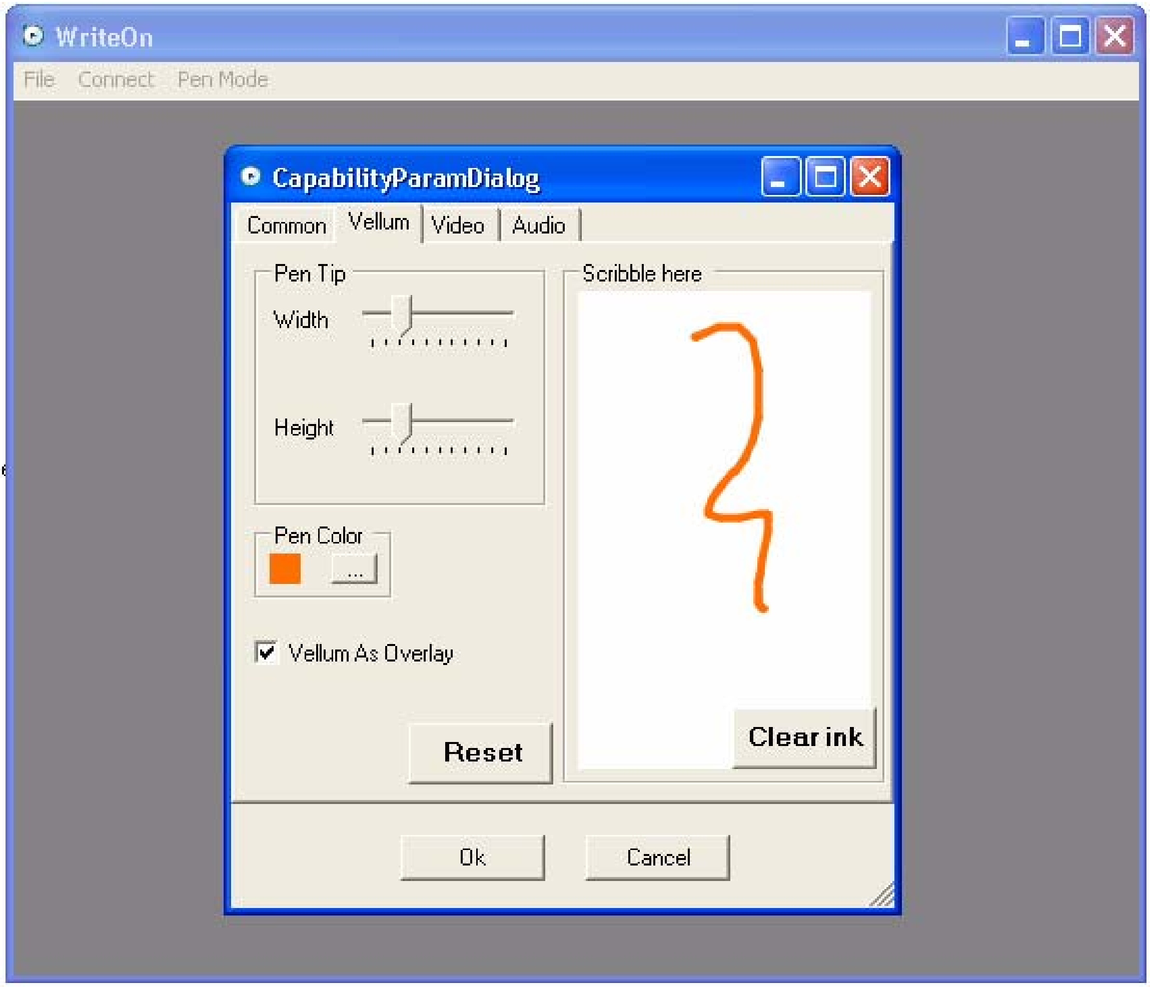


Figure 12: Tray menu options of the WriteOn tool

To create different types of vellum and to change the pen options such as ink color, width, select the *File* => *Properties* => *Vellum* tab menu item. The program will show the above mentioned options making them available to change. See Figure 13.



☑ Vellum As Overlay

Figure 13: Properties dialog box showing vellum properties

The instructor can scribble on the scribble area to test the affect of changing the pen options and can at anytime reset to the previously set values by clicking the “Reset” button. On selecting the *Vellum as overlay* checkbox as in Figure 13, the tool creates an Overlay Vellum. If the box is left unchecked the program creates a Solid Vellum. The default vellum when the WriteOn starts is Overlay Vellum. Clicking *Clear Ink* clears the ink on the scribble area.

Once the vellum is virtually displayed over all of the running applications windows, the *Show Vellum* option automatically changes to the *Pierce Vellum* option in the menu. Now by pulling up this menu, the instructor can switch to the pierce mode of the vellum which will hide the vellum. The menu will toggle from *Pierce Vellum* to *Show Vellum* in order to offer the opposite vellum state on the menu. The instructor can erase the ink annotations on the vellum by inverting the stylus and using the eraser end, if available with their tablet PC hardware. This will change the cursor icon from the pen tip icon to the rectangular eraser icon and erasing can take place with simple gesture movements.

Figure 14 shows example vellum with instructor’s annotations on top of a LogicWorks simulation.

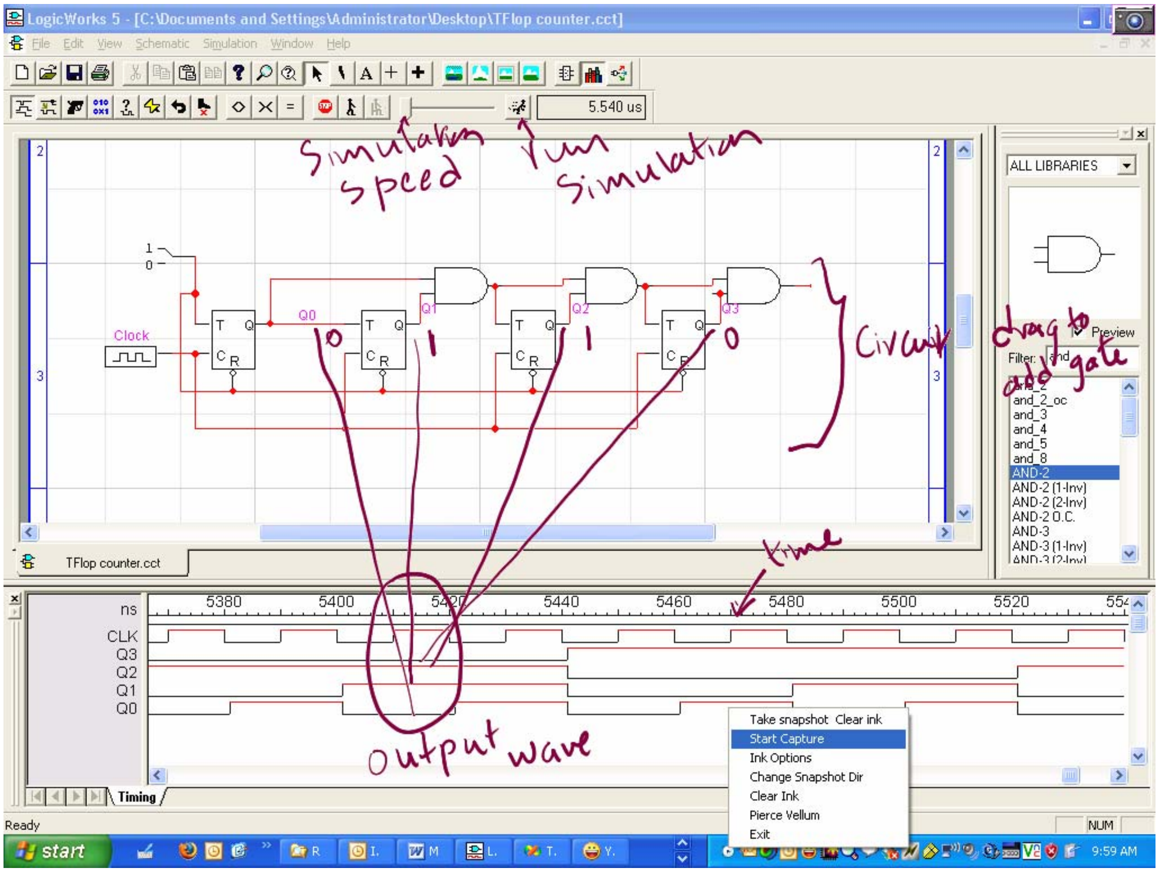
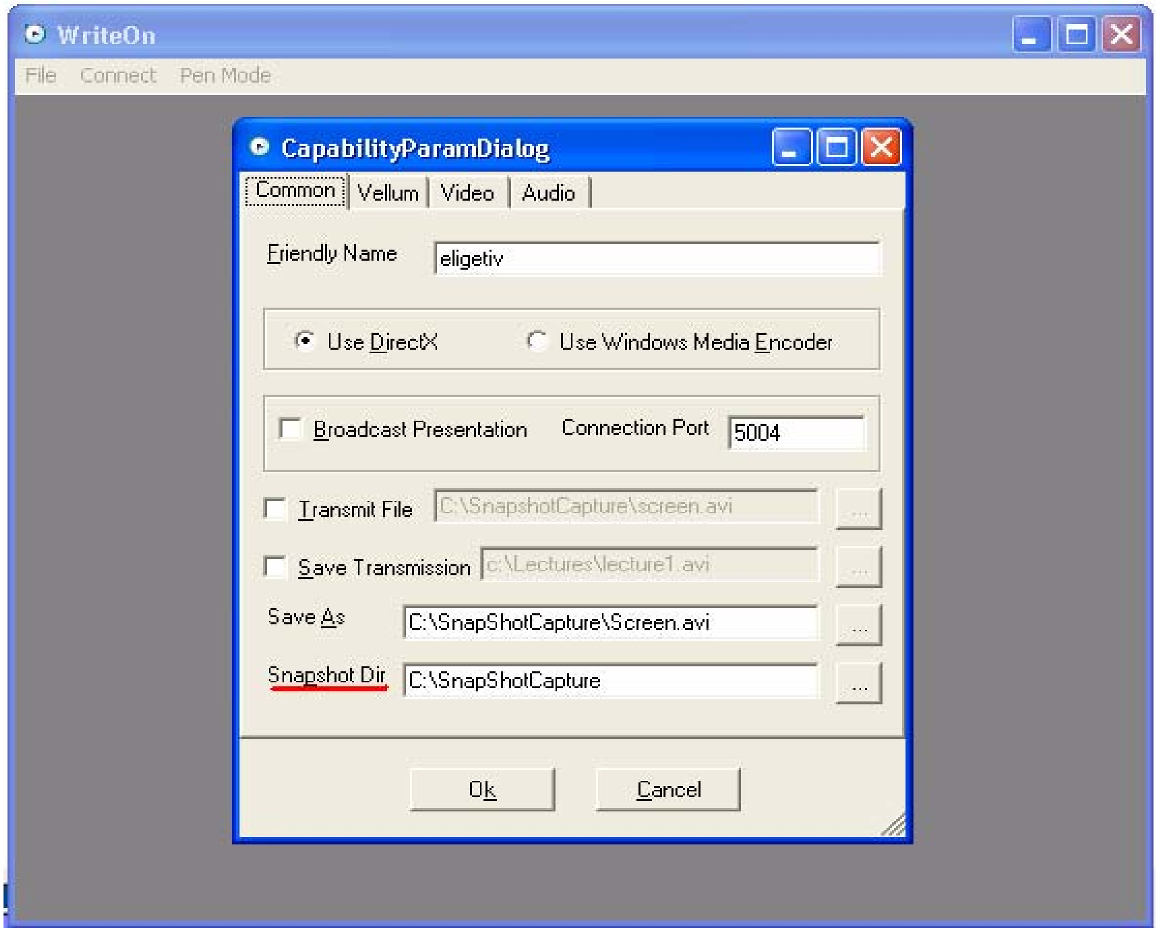


Figure 14: Annotations over Logic Works IDE.

To clear all ink annotations *Clear Ink* option, one of the options appearing on the bottom right corner of Figure 14, is available in the WriteOn tool tray bar menu options. The small camera icon (take snapshot option), which is at the top right corner of the vellum, allows taking a snapshot of the desktop screen and stores the image in the directory mentioned in the *Screenshot Dir* text field of the *File => Properties => Common* tab menu as shown in Figure 15. However, when the instructor first clicks the camera icon, the WriteOn tool pops up a file dialog box to select the directory in which the instructor wants to store the screenshots. *Take Snapshot and Clear Ink* takes the snapshot and also clears the ink annotations on the vellum. Now the eVellum is similar to a new slide to write on for a transparency projector. The instructor can change the snapshot directory by selecting the *Change Snapshot Directory* in the tray bar menu item of the WriteOn tool. If the instructor wants to change the pen options in the middle of the lecture, since the main form is not visible when the vellum is created, s/he has to choose the *Ink Options* menu item which appears on the right click of the WriteOn tray bar icon.



Friendly Name eligetiv

● Use DirectX ○ Use Windows Media Encoder

□ Broadcast Presentation Connection Port 5004

□ Transmit File C:\SnapshotCapture\screen.avi

□ Save Transmission c:\Lectures\lecture1.avi

Save As C:\SnapShotCapture\Screen.avi

Snapshot Dir C:\SnapShotCapture

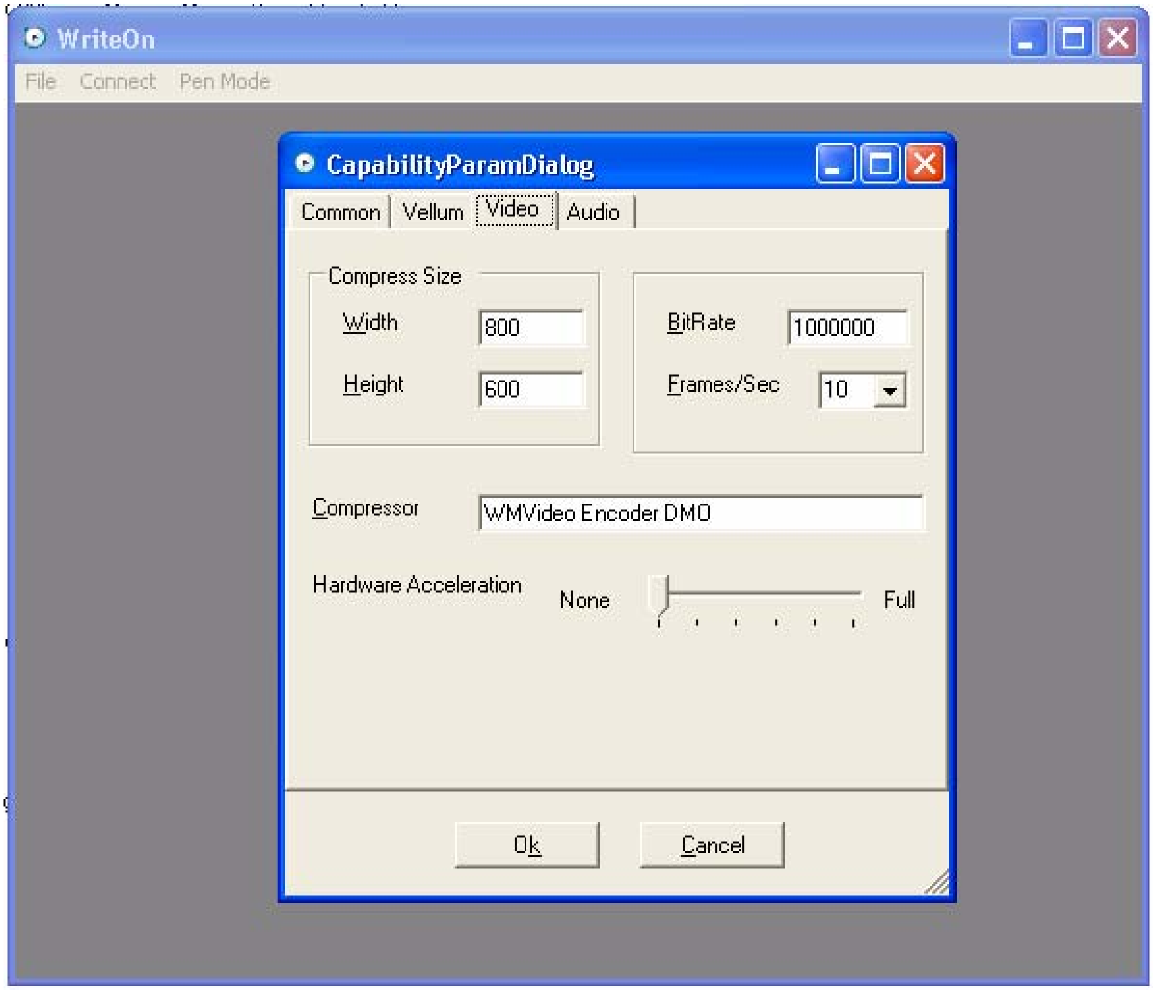
Figure 15: Common properties

8.1.2 Setup for capturing and broadcasting desktop screen activity

As mentioned earlier, the instructor’s display screen may be broadcast to client machines in a multicast sub-network so that each individual student may collect their own version of the screen activity presented by the instructor. To broadcast screen activity, the instructor must first select which technology (DirectX or Windows Media Encoder) to use for capturing the presentation. This option is available under *File* => *Properties* => *Common* menu. Implications of this choice and the details of sub-options are described in the next sections. Next, the instructor must enter the file name of the movie file to be generated from the capture operation and enter this name in the *Save As* text field as shown in Figure 15. The instructor can browse through the directory structure of the local file system to select the path of the movie file name by clicking the browse button (button with three dots) right beside the *Save As* option.

8.1.2.1 Video properties

For capturing and saving to a movie file, the instructor first needs to set the video and audio options. The video options are available under *File* => *Properties* => *Video* tab menu shown in Figure 16. Default values are set to achieve high quality, but the user may adjust these values to suit their needs.



Width 800

Height 600

BitRate 1000000

Frames/Sec 10

Compressor WMVideo Encoder DMO

Figure 16: Video properties

The width and the height options under “Video” tab of the properties dialog box represent the display device screen width and height in pixels. These should be set to the monitor screen resolution width and height for better quality of the movie to be generated. The width and height can also be set to lower resolutions but that will decrease the quality of the captured movie.

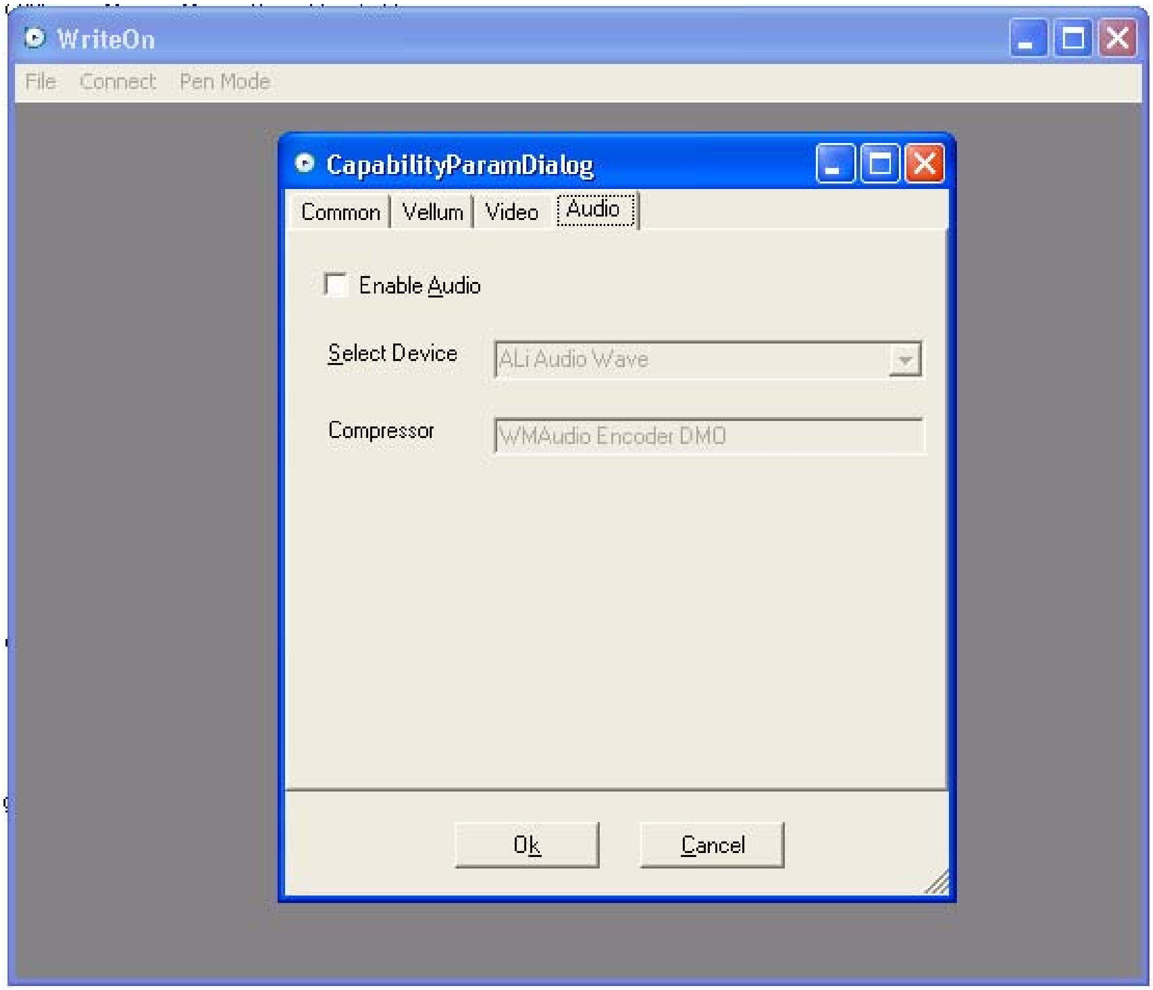
The default video compressor is the WMVideo Encode DMO is used for compressing the captured desktop screen image samples. To achieve better compressor quality the user may install other third party vendors’ compression filters by specifying the name of the compression filter in the *Compressor* text field. Many compressor filters are installed with DirectX installation which is installed when Windows OS is installed. The field *BitRate* shown in Figure 16 denotes the number of bits per seconds. This

BitRate is the rate at with the compressor filters produces the bits per second. The parameter F*rames/sec* indicates the number of frames that the capture filter takes snapshots of the desktop screen. The more number of Frames/sec the more number of screen snapshots the capture filter produces. In effect the video generated with large frames/sec has large size and more CPU time is taken by the capture filter and the compressor filter in processing the video frames. And more CPU cycles are used if higher screen resolutions are used. In general for smaller resolutions higher frames/sec is used and for large screen resolution lesser frames/sec is used. In WriteOn case this value can be ranged from 1 to 10. This is also the optimal playback rate for the video generated by this setting of the filter.

The user can set the Hardware Acceleration to different level to speed up the capturing process. The default value is None (level 5), which means disable Hardware Acceleration (HA) entirely as shown in Figure 16. When capturing starts, the WriteOn tool will automatically set the selected HA level inside the operating system parameters. Normally the HA of the primary display device is set to Full i.e., 0. In Windows XP the HA option is available by right-clicking on the desktop and selecting the *Properties* menu and then going to *Settings* => *Advanced* which opens the display device properties. In the *Troubleshoot* tab menu item there is a HA level bar which can be set. Under normal circumstances, the HA level set at the operating system level is Full in order to take advantage of hardware built into the tablet PC. However, in order to produce a smooth and continuous capturing of the desktop screen, WriteOn sets the HA to None for the duration of capturing. Upon normal termination of WriteOn, the program will automatically restore the previous setting of the HA at the operating system level.

8.1.2.2 Audio properties

Similar to the video properties discussed above, the audio properties are available for adjustment under the *Audio* tab menu of the properties dialog box shown in Figure 17.



☐ Enable Audio

Select Device: ALi Audio Wave

Compressor: WMAudio Encoder DMO

Figure 17: Audio properties

Check *Enable Audio*, shown in Figure 17, to capture the voice of the instructor through the system audio device(s). This will enable the audio devices that are attached to the system and begin collecting and compressing an audio file once the Start Capture command is given.

This menu also allows the user to choose which system audio device is to be used for capture. The choice of audio compressor is also allowed if the user enters the name of an available audio compressor in the filed marked Compressor.

Once the audio and video options are entered, the instructor should click *Start Capture* option under File menu item. This command first sets the hardware acceleration (the user will notice that the screen goes blank for a second) and then starts capturing. The user will know capturing is occurring by the blinking of the mouse pointer on the screen. The menu option *Start Capture* toggles to the *Stop Capture* option to allow the user to subsequently stop capturing when so commanded.

8.1.3 Broadcast of the instructor presentation

To broadcast the presentation, the instructor has to select the *Broadcast Presentation* check box of the *Common* tab menu in the Properties dialog box of Figure 15. Our experiments thus far have shown that DirectX produces the better results in broadcasting the instructor presentation since the technology can captures the overlays and the time lag of the broadcast stream is also less. Therefore, DirectX is the default technology for broadcasting. Although audio service is not available using DirectX technology, one can use WMEncoder if audio broadcast is required.

To broadcast the presentation using DirectX, the instructor must connect to one of the venues which are available under the *Connect* menu item of the main form shown in Figure 11. Either of these two venues are multicast venues. The *Disconnect* option under the *Connect* menu item is used to exit from the multicast venue. Once the WriteOn tool connects properly to a venue, it will show the details of the connection including instructor's friendly name, the venue IP address, and so forth on the title bar of the main form. Now the instructor can broadcast the presentation by clicking the *Start Capture* option again and click the *Stop Capture* to stop broadcast.

In the case of using the WMEncoder technology, the instructor has to simply give the port number in order to broadcast the presentation. The default value of the port number is 5004. Selecting *Start Capture* will broadcast the instructor's presentation if the Broadcast Presentation box is checked.

See Section 6.4 about the DirectX's DirectShow and WMEncoder technologies comparisons and when to use which technology.

8.2 Student usage of the WriteOn tool

To view the instructor's presentation, the students first need to start the WriteOn tool. However, students should know prior to the class as which video compressor technology their instructor is using to broadcast. If the instructor uses DirectShow to broadcast the presentation, then the students need to connect to the same venue as the instructor. Once the presentation is in progress, then the students will see a window which shows the ongoing presentation in case of DirectX's DirectShow. Students can maximize/minimize and full screen the presentation window start annotating on the presentation window as well. To save the presentation, the student has to right click on the presentation window and select the *Save* option. This will save the file in the location entered in the *Save Transmission* text field of the *Common* tab menu, see Figure 15, in the properties dialog box.

If the instructor uses WMEncoder to broadcast the presentation, the students have to select the *Play Online* menu item of the *Connect* menu item of the main form. Upon doing this, the WriteOn tool pops up a URL dialog box to enter the IP address and the port number at which the instructor is broadcasting the presentation. Once the *Go* button is clicked, the tool will show the window which plays the presentation of the instructor. Ink annotating and saving the presentation by the students are not supported in this case.

Chapter 9

Results

WriteOn tool has been tested to assess its usability and the importance of its use in enhancing the learning process in the engineering classroom. Though the tests were not yet conducted extensively over a large number of different types of classroom presentations, the end results of the tests thus far showed that WriteOn can be used to help the instructor better explain the subject and provide the students with a better understanding of the lecture material.

There were three types of tests conducted in evaluating the WriteOn tool. These were: 1) the instructor used only the vellum functionality, 2) the instructor used both the vellum and the capture of the desktop screen of his/her machine, and 3) the instructor used vellum with broadcast functionality and students viewing the instructor's broadcast presentation.

In the first type of experiment, the WriteOn tool is installed on the instructor's machine. The instructor conducted the lecture on electronic circuit design course to the students in an electrical engineering class. The transparency feature of the WriteOn tool is used extensively in explaining the circuit design to the students. Since the vellum resides on top of all applications, the instructors could continuously annotate on the vellum without any interruption. Whenever the instructor needed to switch to the underlying tool which was the focus of the presentation, the switch was made by disabling the vellum by pressing function key F2. Also the instructor used various pens (combinations of colors and width of the ink annotations) to explain several aspects of the circuit design such as circuit elements - resistors, capacitors, currents, links between the nodes, voltage, etc. Students could more readily follow the presentation because of the ability to highlight various aspects of the presentation material through the use of the features noted above. The instructor took several snapshots of the presentation lecture, stored them on the presentation computer, and later distributed to the students later. Students indicated that they thought that this process would help them in better understanding of the material presented when it came time to review the notes outside of class.

The instructor also used WriteOn tool to explain the digital logic simulation using LogicWorks. LogicWorks IDE is used to simulate digital logic by first entering the schematic into the simulator drawing area, entering simulation parameters, and then subsequently running the simulation. Students indicated that they felt that the vellum really provided a value in assisting the instructor in explaining the dynamic nature of the simulation process. They could see the ink annotations of the instructor on the actual operating simulation rather than on still images of snapshots of the simulation in operation. Figure 14 shows the instructor's annotations on the LogicWorks simulation. Students could understand the circuit and the working of the LogicWorks easier because of the transparency feature of the WriteOn tool.

In the second type of experiment, the instructor used the WriteOn tool not only to annotate on the desktop but also to capture the desktop screen activity. The same instructional scenario as in the previous experiment was used and the whole classroom presentation was saved as a movie file. Since the WriteOn tool was capturing the desktop screen, there was some noticeable delay in opening the applications required for the presentation along with some delays in ink appearance during annotations. However, the delay was not distracting and better than other alternatives. This experiment also produced the same student satisfaction effect as in the first experiment. Furthermore, the saved movie file on the instructor's machine was distributed to students in order for them to review the presentation afterward. Again, they indicated that the ability to review the material helped them in understanding the lecture.

Lastly, the students are connected to the instructor's machine using the WriteOn tool and wireless Internet connections. This experiment, however, was conducted with only three students. The students’ and the instructor’s machines are connected to a single wireless router using the 54 Mbps channel. The advantage of this mode of operation is that students could annotate on the window which shows the instructor broadcasts providing them with the opportunity to not only collect the instructor’s notes, but to add their own notes to the presentation. The students were also able to save the whole presentation including instructor's annotations and their annotations to a movie file. Though there is a small delay in transmission of the video stream of the presentation from the instructor's machine to the students' machines of about 3 to 4 seconds, it really did not cause any major problem for the students in understanding the lecture. The lag is because of the loss of video frames sent from the instructor's in the wireless network.

On the whole the above experiments showed that the WriteOn tool can be used to improve classroom presentations. Student satisfaction with the tool was high though there were strong suggestions that eliminating the delays currently in the system would greatly improve the system.

Chapter 10

Conclusions and Future Work

10.1 Conclusions

This thesis project has successfully demonstrated the first core implementation of the WriteOn tool. The tool can be installed as a stand alone software package in a Tablet PC and operates simultaneously with any other Windows application. For proper operation, the target machines should have the ConferenceXP filters and the Windows Media Encoder installed. Usability tests conducted by deploying the WriteOn tool in a classroom scenario showed reasonable and expected results.

The instructors really liked the functionality of annotating on any application as well as the reduction in time they needed to prepare the lecture presentation. Since the WriteOn tool is developed to work on mobile computers, instructors benefited from the ability to move around the lecture room freely. Instructors liked several functionalities that the WriteOn tool provided. These features of the WriteOn tool helped them to improve the classroom presentations. Students also liked the feature that gives them the ability to view the presentation actively and write their notes directly on the presentation, because the reduction in time here gave them more time to focus their efforts on understanding and assimilating the informational aspects of the lecture. Although the preliminary development of the WriteOn tool was successful and the tool behaved as intended, there are several aspects of the tool that can be improved based on the results collected from its the classroom deployment. Some of them are highlighted below:

- The instructor has to set the width, height, frames per sec, and other properties before capturing. It might be difficult for instructors other than those in the CS or

ECE departments to comprehend what these values are, the tool can be simplified to take default values, as in case of audio properties.

- The instructor needs to set the properties every time he/she uses the WriteOn tool for a new presentation.

- The instructors, prior to the classroom presentation, have to test which type of eVellum they need to use for the presentation. Since there are two vellums, which work in different cases, the instructor has to set this value so that the WriteOn tool knows which type of vellum to create. May be the WriteOn tool should allow the use of both types of vellum.

- When the instructor right clicks the tray bar icon of the WriteOn tool, s/he cannot use a stylus immediately to activate the menu items. This delay is irritating for the instructor and sometimes and may not work if the right click is not done properly.

- Students experienced some time lag (of about 4 to 7 seconds) to see the exact same presentation screen as that of the instructor.

Event with these shortcomings, the WriteOn tool never hampered the classroom presentation and passed the initial tests.

10.2 Future Work

There are several aspects that need are to be improved to make the WriteOn tool really ready to be used as an active lecturing tool. One of them is to come up with GUI improvements so that the changes will help the instructor perform the tasks easily. GUI layout similar to the Snipping Tool® or Microsoft’s Ink Art® could be used for the WriteOn tool also. Performance issues, including the capturing process, which needs to be faster, decreasing the size of the generated movie file and the delay at the student end needs to be reduced further. One interesting thing is that students want to submit their slides (with ink annotations) to the instructor during presentation so that instructor can view the slide and discuss students’ input and questions. Therefore, allowing student submissions as a feature will certainly aid in improving the interaction between the instructor and the students. Also the ability to broadcast the voice of the instructor is another feature that needs to be incorporated into the WriteOn tool feature list. Lastly, there is the issue of user properties storage. Whenever, the instructor sets the properties, the WriteOn tool has to store these values in the local machine so that these values can be used for the next presentation. Incorporating the above mentioned features will certainly improve the classroom interaction and learning processing in a classroom scenario.

References

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

2. Steve Wolfman, *Understanding and Promoting Interaction in the Classroom Through Computer-Mediated Communication in the Classroom Presenter Syste*, Ph.D. Thesis, University of Washington, August, 2004.

3. 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

4. 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.

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

6. TechSmith Corporation website, Expertise on capturing tools, codec and media development such as SnagIt, Camtasia Studio, Morae etc, *http://www.techsmith.com/,* founded by William Hamilton in 1987

7. Introduction to Common Object Model(COM) interop, Creating and using COM objects, *http://msdn.microsoft.com/library/default.asp?url=/library/en-*

us/vbcn7/html/vaconCOMInteroperability.asp, Microsoft Corporation, MSDN online library

8. Website describes about .NET Framework SDK tools used to develop, create and run .NET applications, MSDN online documentation, Microsoft Corporation, *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpconNETFrameworkTools.asp*

9. Windows OS Graphical Desgin Interface (GDI) functions, Bitmap functions, *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdi/bitmaps_87eb.asp*, MSDN online documentation, Microsoft Corporation,

10. Classroom Presenter website, Product and development information about Classroom Presenter, DeckBuilder and PowerPoint Plugin, *http://www.cs.washington.edu/education/dl/presenter/*, Computer Science & Engineering, University of Washington website.

11. Official website of DyKnow corporation, Expertise in teaching tools such as DyKnow Vision, DyKnow Monitor, *http://dyknow.com*, founded by Dr. Dave Berque,P.h.D. Professor of Computer Science, DePauw University

12. Webpage about Microsoft's OneNote 2003 product information, *http://www.microsoft.com/office/onenote/prodinfo/default.mspx*, OneNote makes a better way to capture, organize, find, and share notes, Microsoft Corporation.

13. 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

14. Richard Anderson, Ruth Anderson, Beth Simon, Steve Wolfman, Tammy VanDeGrift and Ken Yasuhara, *Experiences with a Tablet PC Based Lecture Presentation System in Computer Science Courses,* Special Interest Group on Computer Science Education (SIGCSE), 2004

15. 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.

16. Microsoft's Experience Pack for Tablet PC, Product, installation and usage information about the experience pack tools, includes ink tools like Snipping tool, Ink Art, Ink Crossword, Ink desktop, Media transfer, Energy Blue, *http://www.microsoft.com/windowsxp/downloads/tabletpc/experiencepack/default.mspx*, Microsoft Corporation

17. Website about Microsoft .NET Framework developer center, .NET runtime engine (CLR) installation, .NET general user information, *http://msdn.microsoft.com/netframework/downloads/framework1_1redist/*

18. Microsoft Visual Studio 2005 production information, *http://msdn.microsoft.com/vstudio/default.aspx*, Microsoft Corporation

19. Microsoft ConferenceXP research official website, Product and source code information about ConferenceXP client, ConferenceXP message boards, *http://www.conferencexp.com/community/default.aspx*, Microsoft Corporation

20. Jesse Liberty, Programming C#, O'reilly publications, 4th Edition, 2005

Appendix I: Registering system wide keyboard hooks

This appendix describes the source code used to hook into the Windows OS. The InstallHook() function of the HookManager hooks to Windows keyboard event queue. The InstallHook() registers KeyBoardDistiller() function by passing the handle to the SetWindowsHooksEx() Win API. By this whenever a keyboard event occurs KeyBoardDistiller() function is called which in-turn checks for the desired key and notifies the delegator.

```
/// <summary>
/// A system wide HookManager which hooks for a particular key.
/// When that key is pressed it will gives a callback to
/// FunctionKeyPressed event handler
/// </summary>
public class HookManager
{

    /// <summary>
    /// function used to register the hooks
    /// </summary>
    /// <param name="code">type of hook</param>
    /// <param name="func">callback function pointer</param>
    /// <param name="hInstance">Handle to the DLL hook </param>
    /// <param name="threadID">thread id</param>
    /// <returns></returns>
    [DllImport("user32.dll", CharSet=CharSet.Auto,
             CallingConvention=CallingConvention.StdCall)]
    public static extern int SetWindowsHookEx(int code,
            HookProc func, IntPtr hInstance, int threadID);

    /// <summary>
    /// function used to uninstall registered hooks
    /// </summary>
    /// <param name="hhook"></param>
    /// <returns></returns>
    [DllImport("user32.dll",CharSet=CharSet.Auto,
             CallingConvention=CallingConvention.StdCall)]
    public static extern bool UnhookWindowsHookEx(int hhook);

    /// <summary>
    /// function used to call next hook when a keyboard event
    /// </summary>
    /// <param name="hhook"></param>
    /// <param name="code"></param>
    /// <param name="wParam"></param>
    /// <param name="lParam"></param>
    /// <returns></returns>
    [DllImport("user32.dll",CharSet=CharSet.Auto,
             CallingConvention=CallingConvention.StdCall)]
    public static extern int CallNextHookEx(int hhook,
            int code, Int32 wParam, IntPtr lParam);

public delegate int HookProc(int code, Int32 wParam, IntPtr lParam);

    /// <summary>
    /// event used to fire when the desired key is pressed
    /// </summary>
    public event DesiredFunctionKeyPressed FunctionKeyPressed;
    public delegate void DesiredFunctionKeyPressed();

    [StructLayout(LayoutKind.Sequential)]
    public class LPARAM
    {
        public int vkCode,scanCode,flags,time,dwExtraInfo;
    }

    /// hook types
    public enum HookType : int
    {
        WH_KEYBOARD = 2,
        WH_KEYBOARD_LL = 13,
        WH_MOUSE = 7,
        WH_MOUSE_LL = 14
    }

    private const int WM_KEYUP              = 0x101;
    private const int WM_SYSKEYUP           = 0x105;

    private int funHook = 0;
    private Keys desiredKey = Keys.F2;

    HookProc keyboardHookProc;

    public bool InstallHook()
    {
        keyboardHookProc = new HookProc(this.KeyBoardDistiller);
        // installing the hook
        funHook = SetWindowsHookEx( (int)
HookType.WH_KEYBOARD_LL,
keyboardHookProc,Marshal.GetHINSTANCE(Assembly.GetExecutingAssembly()
.GetModules()[0]), 0);

                // if funHook is zero then hook is not registered
                if(funHook == 0)
                        return false;
                return true;
        }

        private IntPtr Handle;
        public void SetHandle(IntPtr handle)
        {
                this.Handle = handle;
        }

        /// <summary>
        /// key used to fire the DesiredFunctionKeyPressed event
        /// </summary>
        public Keys Key
        {
                get { return desiredKey; }
                set { desiredKey = value; }
        }

        bool altTab = true;
        public bool EnableAltTab
        {
                get { return altTab; }
                set { altTab = value; }
        }
```

```
    public void UnInstallHook()
    {
        // uninstalling the hook
        UnhookWindowsHookEx(funHook);
        funHook = 0;
    }

    private int KeyBoardDistiller(int code, Int32 wParam, IntPtr lParam)
    {
        int keyState = (int) wParam;
        try
        {
            if (code >= 0)
            {
                // check the key pressed event message
    if(keyState == WM_KEYUP || keyState == WM_SYSKEYUP)
                {
                    LPARAM keyEvent = (LPARAM)
Marshal.PtrToStructure( lParam, typeof(LPARAM));

                    Keys key = (Keys) keyEvent.vkCode;
            // check for the key if it is the one which has to be signaled
                    if(key.Equals(desiredKey))
                    {
                        if(FunctionKeyPressed != null)
                            FunctionKeyPressed();
                    }
                    else if(altTab)
                    {
                        KeyEventArgs ke = new KeyEventArgs(
key );
```

```
                        }
                    }
                }
            }
            catch (Exception) {}

        return CallNextHookEx(funHook, code, wParam, lParam);
        }
    }
```

Appendix II: Functions used to take snapshot of the desktop screen

Appendix II describes about the source code used to take the snapshot of the desktop screen. The main function is *TakeSnapshot()* which takes a desktop window handle for capturing. It then prompts the user for the first time to the directory location to where the snapshot should be saved. This is done by calling *SetSnapshotDir()*. *BitBlt()* function is a Windows GDI function to get the bitmap information of the desktop and store it as a image object. At the end the image is saved to the destination directory.

```
    public class ScreenSnapshotTaker
    {

    // gets the device context of the given window handle
        [DllImport("user32.dll")]
        private static extern IntPtr GetWindowDC(IntPtr hwnd);

        // gets the desktop window handle
        [DllImport("user32.dll")]
        private static extern IntPtr GetDesktopWindow();

        // gets the client rectangular region of the given window handle
        [DllImport("user32.dll")]
        private static extern bool GetClientRect(IntPtr hwnd, ref
RECTANGLE_STRUCT rectangle);

        static int SRCCOPY = 0x00CC0020;  // dest = source
        static  int CAPTUREBLT = 0x40000000;
```

```
/// <summary>
/// copes the data from the hSrcDc device context to the hDestDC
/// within the bounds specified
/// </summary>
/// <param name="hDestDC"> destination device context</param>
/// <param name="x"> X position of the destination DC </param>
/// <param name="y">Y position of the destination DC</param>
/// <param name="nWidth"> width of the destinatin DC</param>
/// <param name="nHeight"> height of the destination DC</param>
/// <param name="hSrcDC"> source device context</param>
/// <param name="xSrc">X position of the source DC</param>
/// <param name="ySrc">X position of the source DC </param>
/// <param name="dwRop">raster operation code</param>
/// <returns></returns>
[DllImport("gdi32.dll")]
private static extern UInt64 BitBlt(IntPtr hDestDC, int x, int y,
                              int nWidth, int nHeight, IntPtr hSrcDC,
        int xSrc, int ySrc, System.Int32 dwRop);

public struct RECTANGLE_STRUCT
{
        public int left;
        public int top;
        public int right;
        public int bottom;

        public int Width
        {
                get { return right - left; }
        }
        public int Height
        {
            get { return bottom - top;  }
        }
    }

    public ScreenSnapshotTaker()
    {
    }

    string snapshotDir = null;
    bool isDirSet = false;

    public string SnapShotDirectory
    {
        get { return snapshotDir; }
        set
        {
            if(!isDirSet && snapshotDir != null &&
!snapshotDir.Trim().ToLower().Equals(value.Trim().ToLower()))
                isDirSet = true;
            snapshotDir = value;
            count = 1;
        }
    }

    // the function which takes the snapshot of
      // the given window handle
    public void TakeSnapshot(IntPtr handle)
    {
        RECTANGLE_STRUCT clientRect = new
RECTANGLE_STRUCT();
        if(handle == IntPtr.Zero)
        {
            clientRect.top = 0;
            clientRect.left = 0;
            clientRect.right = Screen.PrimaryScreen.Bounds.Width;
            clientRect.bottom=Screen.PrimaryScreen.Bounds.Height;
        } else
            GetClientRect(handle, ref clientRect);
        Image desktopImage=new Bitmap(clientRect.Width,
clientRect.Height);
        Graphics gr1 = Graphics.FromImage(desktopImage);
        IntPtr dc1 = gr1.GetHdc();
        IntPtr dc2;
        // if the handle is zero then capture the desktop window
        if(handle == IntPtr.Zero)
            dc2 = GetWindowDC(GetDesktopWindow());
        else
            dc2 = GetWindowDC(handle);
        BitBlt(dc1, 0, 0, clientRect.Width, clientRect.Height, dc2, 0, 0,
SRCCOPY | CAPTUREBLT);
        gr1.ReleaseHdc(dc1);
        gr1.Dispose();
        if(!isDirSet)
            SetSnapshotDir();
        isDirSet = true;
        desktopImage.Save(snapshotDir + "\\"+ prefix + "_" + count +
".jpg",
                System.Drawing.Imaging.ImageFormat.Jpeg);
        count++;
    }
```

```
private string prefix = "LectureSlide.jpg";
private int count = 1;

// function to change the snapshot directory
public void SetSnapshotDir()
{
        SaveFileDialog saveAsDlg = new SaveFileDialog();
        saveAsDlg.InitialDirectory = snapshotDir;
        saveAsDlg.FileName = prefix;
        saveAsDlg.DefaultExt = "jpg";
        saveAsDlg.Filter = "JPEG Files (*.JPG)|*.JPG";
        saveAsDlg.ValidateNames = true;
        saveAsDlg.ShowDialog();

        string fileName = saveAsDlg.FileName;
        int startIndex = fileName.LastIndexOf('\\') + 1;
        int endIndex = fileName.LastIndexOf('.');
        if(endIndex == -1)
                endIndex = fileName.Length;
        prefix = fileName.Substring(startIndex, endIndex - startIndex);
        string parentDir = new FileInfo(fileName).Directory. FullName;

        if(!snapshotDir.Trim().ToLower().Equals(parentDir.ToLower()))
        if(SnapshotDirChange != null)
                {
        // notify the listeners that snapshot directory is changed
                        SnapshotDirChange(parentDir);
                }
        snapshotDir = parentDir;
        count = 1;
        }

        public event SnapshotDirChangeDelegator SnapshotDirChange;

        public delegate void SnapshotDirChangeDelegator(string fileName);

        private void fileDialog_FileOk(object sender,
System.ComponentModel.CancelEventArgs e)
        {
                e.Cancel = false;
        }
    }
```

Various Bitmap functions used in SnapshotTaker class is explained in [9] link.

Appendix III: Capture filter IDL and steps to generate COM dll

This appendix describes the Intermediate Definition Language (idl) file used to create the COM wrapper around the capture filter. It also describes the steps involved in creating the COM dll.

IScreenCapFilter.idl interface

```
import "oaidl.idl";
import "ocidl.idl";
[
    object,
    uuid(F4EB785E-8205-423a-9ED4-03572492CFE5),
    nonextensible,
    pointer_default(unique)
]
interface IScreenCapFilter : IUnknown
{
    [id(1), helpstring("method Width")] HRESULT Width([in] int width);

    [id(2), helpstring("method Height")] HRESULT Height([in] int height);

    [id(3), helpstring("method FramesPerSec")] HRESULT FPS([in] int fps);

    [id(4), helpstring("method SetWindowHandle")] HRESULT SetWindowHandle([in] int winHndl);

    [id(5), helpstring("method ReleaseResources")] HRESULT ReleaseResources();
};

[
    uuid(64290030-E89E-43f7-9E12-E359AD17CE5E),
    helpstring("IScreenCapFilter")
]
coclass CaptureFilter
{
    [default] interface IScreenCapFilter;
};
```

Filters.idl interface to generate the COM dll

```
[
    uuid(4D2AD7BC-11E0-4224-988D-2F0876709EA9),
    version(1.0),
    helpstring("Managed Interface for the Capture filter"),
    helpcontext(00000000)
]
library CaptureFilterDll
{
   #include "IScreenCapFilter.idl"
}
```

The *IScreenCapFilter.idl* file is used to generate the necessary header files and interfaces for the *CaptureFilter* to implement. An idl file just contains the type definitions and the declarations of the interface functions that are present. These types are defined according to the Common Language Specification (CLS) types of the .NET framework for interoperability among different languages. This means the functions are just definitions and actual implementations are done in the respective language. On compiling the idl file several header files are generated which contains the implementations of the interface exposed. The implementation just transforms the incoming function calls and makes the function calls to the actual object which implements them. The dynamic link library (dll) generated out of the whole process (described in the following section) is used by the WriteOn tool to make function calls to the *CaptureFilter*. Basically the dll contains the COM interfaces exposed so that when a function is called, the .NET runtime engine will give a call back to the actual object (*CaptureFilter*) which implements the COM interfaces.

Looking at the *IScreenCapFilter.idl* more closely, reveals *uuid(F4EB785E-8205-423a-9ED4-03572492CFE5)* part of the interface is the Universal unique identification number used to identify each type and library in the .NET framework. This will be unique all over the world. Therefore *uuid(F4EB785E-8205-423a-9ED4-03572492CFE5)* is the id of the *IScreenCapFilter* interface. Since *IScreenCapFilter* is a COM interface, it has to extend and implement the *IUnknown* COM interface. All the function declarations within the body of the *IScreenCapFilter* are the functions which the *CaptureFilter* implements. The first method declaration of the *IScreenCapFilter.idl* is:

```
[id(1), helpstring("method Width")] HRESULT Width([in] int width);
```

The parameter passed is of type “int” and before that it is qualified as “[in]” which means the parameter is passed by value to the COM object. Whatever changes are made to this parameter do not affect the actual parameter that is passed. There are 3 types of argument passing in COM. They are:

[in] – passed by value, does not affect the actual value of the parameter passed.

[out] – passed by out, does not take the value of the passed parameter value but when the called function returns, the callee sees the updated value in the passed parameter

[in] [out] – passed by reference, are similar to references in C++ and Java programming language.

The *uuid(64290030-E89E-43f7-9E12-E359AD17CE5E)* of the coclass represents the UUID of the class which implements the *IScreenCapInterface*.

The *Filters.idl* contains the information about the library file to be generated. The *uuid(4D2AD7BC-11E0-4224-988D-2F0876709EA9)* is the id of the library to be generated. When compiled, the MS Visual Studio compiler generates the dll with the file named as CaptureFilterDll.dll. The *#include* part of the library section tells the compiler what interfaces include in the generated dll. [7] goes into detail about how to implement the COM interfaces.

Procedure to generate DLL and the header files.

The following steps were used to create the interoperable wrappers and header files:

1. First, create the library block and include the files whose interfaces has to be exposed as in Filters.idl.
2. Open the Microsoft Visual Studio command prompt.
3. To generate the UUID for the types, enter “guidgen” at the command prompt. This will open a message box showing the new UUID that can be used.
4. Enter the “MIDL” command at the prompt giving the library file idl to create the type library. The generated file contains the <library file name>.tlb. The .tlb file contains the information about the types that should be exposed.
5. Then, enter “TLBIMP” command at the prompt passing the tlb file as the argument. This will create the required interoperable wrapper dll.
6. If the “TLBIMP” does not create the functions with right type information, disassemble the dll file in order to have the correct types. Since this is done with “ILDASM” command, enter “ILDASM” command against the dll to disassemble

it. This will pop up the dialog box showing which format the disassemble file to be generated. Turn on the text format and save the file using File => Dump. Uncheck every option except Dump the IL code. The file generated has the .il (interoperable language) extension.

7. Then manually edit the disassemble file generated in step 6.
8. After completing the editing recompile the IL file using the “ILASM” command.
9. To create the header files, repeat only step 4 but with IScreenCapFilter as the argument.

Thus, the generated interoperable wrapper dll can be added as the reference to the project which requires the dll. For more description about the commands used for generating the interoperable wrappers please look at [8].

Appendix IV: Changing Hardware Acceleration Settings for Graphics Hardware

Windows OS uses hardware acceleration to improve display performance. If using hardware acceleration causes a problem, such as mouse pointer problems or corrupt images, turn off some or all hardware acceleration features. By turning off hardware acceleration, it is possible to control manually the level of acceleration and performance supplied by your graphics hardware, which can help with troubleshooting display problems.

Hardware acceleration for graphics hardware has six settings. Table 9.4 shows the settings and their meanings.

Hardware Acceleration for Graphics Hardware

Setting	**Description**
None -	Disables all accelerations. Use this setting only if your computer frequently stops responding or has other severe problems.
1	Disables all but basic accelerations. User this setting to correct more severe problems.
2	Disables all DirectX® Graphics accelerations, as well as all cursor and advanced drawing accelerations. Use this setting to correct severe problems with DirectX accelerated applications.
3	Disables all cursor and advanced drawing accelerations. Use this setting to correct drawing problems.

4	Disables cursor and bitmap accelerations. Use this setting to troubleshoot mouse pointer problems or corrupt images.
Full	Enables all acceleration features. This setting is recommended if your computer has no problems.

Note

If you use multiple monitors, changing hardware acceleration settings affects all monitors.

To change hardware acceleration

1. Right-click the desktop, and then click *Properties*.
2. In the *Display Properties* dialog box, click the *Settings* tab, and then click the *Advanced* button.
3. Click the *Troubleshoot* tab, and then choose the desired level of hardware acceleration.

Appendix V: Developer's Guide

This section describes about the various aspects as how to setup, to build, to debug and to run the WriteOn tool. The entire application is developed using Microsoft's Visual Studio (MSVS).NET framework IDE. MSVS can be installed from the Microsoft website [18].The WriteOn tool is named as Vellum solution in MSVS. MSVS allows setting up multiple projects at one place called *Solution*. So a solution is a combination of multiple projects either dependent on each other or not. Once the Visual Studio is installed, the Vellum solution has to be opened by double clicking the solution file. MSVS automatically opens the solution and shows the files and projects in the "Solution Explorer". To open a particular file in a project the user has to double click the file icon in the solution explorer of the MSVS. It will open the file in the central workspace area. Several properties of the project, files, resource files and other files can be viewed by right-clicking and then selecting properties popup menu item. The following sections describe how to setup, build and run the Vellum solution.

Setting up Vellum solution

Vellum solution consists of 6 projects. First of all to create a blank solution select the File=>New=>Blank Solution menu item. MSVS prompts for the solution name to be entered. Entire the solution name to be created and click Ok button. This will create a blank solution in MSVS. Then add the projects to the solution by selecting "Add=>New Project" or "Add=>Existing Project" on right-click of the solution icon on the "Solution Explorer". This action adds a project or creates a new project file accordingly. As said there are 6 projects in Vellum Solution. These are:

- CaptureFilter
- DesktopCapture
- HookManager
- RemoteCapabilities
- Transparency and

- Util

The following section describes about the individual projects.

CaptureFilter project

CaptureFilter is a C++ project. This project contains C++ and C files used to create the capture filter. *CaptureFilter.cpp* file in the project is the source code used for capture the desktop screen. This contains the implementation of *CCaptureFilter*, a DirectShow filter component. Capturing of the desktp screen bitmap is performed by calling *CopyScreenToBitmap1()* function. *CCaptureFilter* class implements the *IScreenCapFilter.h* interface which is defined in *IScreenCapFilter.idl*, a COM intermediate definition language file. See Appendix III as how to create the COM wrappers and the header files. CaptureFilter output file is set as *CaptureFilter.ax* file. This means that whenever the CaptureFilter is built it outputs *CaptureFilter.ax* file. To change the output file name go to the properties of the CaptureFilter and then select "Linker => General". In the "Output file" textbox enter a different name with .ax extension. This project depends on the DirectX SDK baseclasses, so in order to compile the files install the DirectX and then point the "Additional Library Directories" (go to CaptureFilter project properties and then select Linker => General) to the include directory of the DirectShow. Also include "/MACHINE:I386 /IGNORE:4089 /IGNORE:4098" line in the "Additional Options" of the Command Line under Linker properties of the project. The above line is added to remove unnecessary warnings shown by the MSVS while building the project.

HookManager project

HookManager project is a C# Class Library project i.e., MSVS outputs a DLL (dynamic link library) when it is built. A class library project is created by selection "Add=>New Project" of a solution properties popup menu item. MSVS shows "Add New Project" dialog box, select "Class Library project" in the "Visual C# projects" list. When Vellum Solution is opened it shows the HookManager in solution explorer. The main purpose of this project is that it installs the keyboard hook into the Windows OS keyboard event queue and notifies the WriteOn tool whenever a specified key is pressed. *HookManager* class is used to perform the above functions. See Appendix I about the working of the *HookManager*. There is no need to setup any other options in this project.

RemoteCapabilities Project

RemoteCapabilities project is also a C# Class Library project. This project contains software classes and modules used to play the broadcasted video samples by the *MonitorDevice*. *MonitorDevice* is just an abstract class which is defined in this project. But the actual implementation is in *DirectXDevice* and *WMEncDevice* classes of the DesktopCapture project. In case of DirectX broadcast, *RemoteCapability* class controls the playing of the broadcast video and the video is played on *RemoteDesktop* form class where as in case of WMEncoder broadcast *WMPlayerForm* class is used to play the broadcast video. RemoteCapabilities Project depends on the number of other dlls like ConferenceXP dlls, DirectShow dlls, Microsoft.Ink SDK dlls. So to build the project the developer has to add these dlls to the project. This can be done by selecting “Add References” of the RemoteCapabilities properties menu item. MSVS then shows “Add Reference” dialog box. Select the necessary dlls by clicking the Browse button and click Ok button. This will add the dlls to the project. Now the project is ready to build

Transparency project

Transparency is also a C# class library project. It contains the software classes used to create different types of vellums. eVellum is an abstract software class called *AbstractTransparentVellum*. Both the *SolidVellum* and *OverlayVellum* classes extend *AbstractTransparentVellum* class to implement the Solid Vellum and Overlay Vellum functionalities. *ChildOverlay* class is the child window which sits on the parent window (*OverlayVellum* class is used to design the parent window). *InkEraser* class handles erasing of the ink strokes and *InkObject* class handles collecting the ink strokes. *ScreenSnapshotTaker* class handles taking the snapshot of the screen, see Appendix II for further details of the internal working of this class. The project depends on the

Microsoft.Ink dll. Add the reference to this dll similar to the way it is done in RemoteCapabilities project.

Util project

Util project is actually presents the Model section of the Model View Controller paradigm. It contains *CaptureParams* class which stores various properties required to capture the screen, play the video, bitrate, width, height, etc. It is also a C# class library project.

DesktopCapture project

DesktopCapture project is the main C# executable project. The output of the project is an executable of the WriteOn tool. So to create a Visual C# executable project, create new project by selecting “Windows Application” of the Visual C# projects instead of the “Class Library” project. DesktopCapture contains source files for capturing the desktop screen using DirectX and WMEncoder. *DirectXDevice* and *WMEncDevice* classes handle the capturing processes. The main form is Form1 class which shows up when WriteOn tool is started. *CaptureController* class handles creation of *MonitorDevice*, eVellum, playing the video, etc. Since the above components are in built in different projects DesktopCapture project depends on all other projects except CaptureFilter project. To add project dependencies to the DesktopCapture project select “Project dependencies” of the DesktopCapture properties menu item. This shows Project dependencies dialog box of a project. Check all the projects which the current projects depends on and then click Ok button. By doing so adds dependencies to a project. The advantage of this is that when the main project gets build all the dependent projects also gets build.

Once setup of the projects in a solution is done, the solution has to be build to generate the executable file. The following section deals how to build a solution or a project

Building the Vellum solution

Whenever, VS solution is opened it will be opened either in Release mode or in Debug mode. When the project is build in Release mode the executable is generated in the <Project Dir>/bin/Release directory (<Project Dir> is the location of the project where it is created). The problem with this mode is that the developer cannot debug in this mode. Generally when the application is fully developed and needs to be run it is changed to Release mode. In order to debug an application one has to set the mode to Debug mode. In debug mode, the developer can debug the application while running. The output build executable file is generated in the <Project Dir>/bin/Debug directory. On the top of the MSVS there is a configuration drop down menu item to toggle between Release and Debug modes.

To build the vellum solution select Build Menu item and say “Build Solution”. This builds the entire solution i.e., all the files of the projects in a solution are compiled. To build individual project, right click the project to be built and select “Build” popup menu item. Generally when lot of source files are modified in different projects “Build Solution” is used otherwise individual project is built. Sometimes projects and solution change very drastically because of adding references or adding other projects or adding source files. In this case it is better to use “Rebuild Solution” since it will build all over again. The difference between Rebuild option and Build option is that the Build option will only compile the modified files but Rebuild option compiles all the files irrespective whether they are modified or not.

After the projects and the solution are built then the entire application is ready to debug or run. Next section describes about how to debug and run the Vellum solution.

Debugging and Running the Vellum Solution

To debug the Vellum Solution set the configuration mode to Debug mode. Then right click the DesktopCapture project and select “Debug => Start New Instance”. This will start the WriteOn tool in debug mode. While debugging the WriteOn, if at any point in time the developer wants to check the status of the objects at a particular line in the source code, one can insert the break point at that line. MSVS pauses the running of the WriteOn at that line. By this developer can check the variable values by selecting “Debug => Windows => Locals”. This will show a small window at the bottom of the MSVS showing the variables and their values at that point. This lets the developer to check the status of an object and if anything wrong in the logic s/he can stop the debugger and restart it after making necessary corrections in the code. If everything is running accordingly press F5 or “Debug => Resume” menu item to resume the execution after the break point. This will continue execution of the WriteOn application. There are other options like StepIn, StepOver, etc which helps the developer perform more debugging operations. Refer MSVS product documentation for further details about these options.

In order to run the application simply go to the DesktopCapture project location and then go to bin/Release and run the DesktopCapture.exe file to run the WriteOn tool.