Columbus State University
CSU ePress

Theses and Dissertations Student Publications

5-2003

Kiosk Engine: A Platform Independent Solution For Multimedia Kiosks

Manish Satish Shah
Columbus State University

Follow this and additional works at: https://csuepress.columbusstate.edu/theses_dissertations

Part of the Computer Sciences Commons

Recommended Citation

Shah, Manish Satish, "Kiosk Engine: A Platform Independent Solution For Multimedia Kiosks" (2003). *Theses and Dissertations*. 63.
https://csuepress.columbusstate.edu/theses_dissertations/63

This Thesis is brought to you for free and open access by the Student Publications at CSU ePress. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of CSU ePress.

KIOSK ENGINE
A Platform Independent Solution For Multimedia Kiosks

Manish Satish Shah

Digitized by the Internet Archive
in 2012 with funding from
LYRASIS Members and Sloan Foundation

http://archive.org/details/kioskengineplatf00shah

Columbus State University

The College of Science

The Graduate Program in Applied Computer Science

Kiosk Engine
A Platform Independent Solution For Multimedia Kiosks

A Thesis in

Applied Computer Science

by

Manish Satish Shah

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2003

© 2003 by Manish Satish Shah

10 21-03

I have submitted this thesis in partial fulfillment of the requirements for the degree of Master of Science.

04/30/03
Date

Manish Satish Shah

We approve the thesis of Manish Satish Shah as presented here.

7/9/03
Date

Stan Kurkovsky
Dr. Stan Kurkovsky, Assistant Professor
of Computer Science, Thesis Advisor

7/9/03 2:25 p.m.
Date

7/9/03
Dr. Bhagyavati, Assistant Professor
of Computer Science

07/09/03
Date

V. Zanev
Dr. Vladimir Zanev, Associate Professor
of Computer Science

Abstract

In today's fast paced life saving time is the first priority of every business organization and presenting information visually is a highly desirable and the easiest way to impressively communicate to people. For that purpose, there are interactive devices called kiosks, which display multimedia data on a display and are quite often attached to other input/output hardware like keyboard, mouse, printer, etc. Most of today's kiosks are custom made and are designed for specific set of target platform. The task of creating a kiosk can turn out to be difficult and fairly time consuming. The aim of this thesis is to research an easier and faster way to develop kiosks and to generalize the processing power behind different kiosks by developing a driver program, which takes some form of input and multimedia data and presents the information as a kiosk.

Putting it to reality, we developed a light-weight engine and called it Kiosk Engine, KE for short. KE is designed to take input in the form of an XML file which specifies the sources of multimedia data such as audio files, video files, text, image files and documents along with some attributes such as size, color, location, order and grouping with respect to other objects on the kiosk. As aimed, *the processing power is separated from data* and hence all that is required to be changed for making different kiosks is the multimedia data and the input XML file. Besides, platform independence is achieved by using Java™ as the programming language. Hence Kiosks driven by Kiosk Engine can run on any Java Virtual Machine [LindholmY03] that has the necessary framework support.

The report shows comparison of KioskEngine with other products on the market, followed by a detailed explanation of XML file structure for KioskEngine and the Java™ classes in KioskEngine. The intent of this thesis is also to demonstrate the capability of Kiosk Engine by developing a working prototype, to research the future prospects of Kiosk Engine and explore additional functionalities that could be incorporated into Kiosk Engine with little extra work.

Table of Contents

Abstract........iii

Table of Contents........v

List of Figures.........vii

List of Tables.........viii

Acknowledgements........ix

1. Introduction........1

2. Motivation........2
  2.1 Desired Features........3

3. Survey of Other Products on the Market........4
  3.1 The Apunix Kiosk Engine for the Java™ Platform........4
  3.2 Kiosk-In-A-Box from Mass Multimedia, Inc........6
  3.3 KioskEngine™ from Touch Controls, Inc.........6
  3.4 Our Product: Kiosk Engine (KE)........8
  3.5 Comparison of Products........9

4. Data Representation in Kiosk Engine........11
  4.1 The Structure of XML file for KE........11

5. Implementation of Kiosk Engine........13
  5.1 Other languages and development tools........13
    5.1.1 Microsoft® Visual Basic 6 and Visual Basic .NET........13
    5.1.2 Macromedia® Flash........14
  5.2 Sun® Java™........15
  5.3 Comparison of Java™ with other languages and tools........18

6. Content Requirements........19
  6.1 Elements of XMLKiosk file for KE........19
    6.1.1 <kiosk> tag........19
    6.1.2 <screen> tag........20
    6.1.3 <header> tag........21
    6.1.4 <text> tag........22
    6.1.5 <img> tag........23
    6.1.6 <action> and <target> tags........24

    - 6.1.7 <textArea> tag 25
    - 6.1.8 <aud> tag 26
    - 6.1.9 <vdo> tag 26
    - 6.1.10 <file> tag 27

7. Program Architecture of Kiosk Engine 29
  - 7.1 Conceptual Classes of Kiosk Engine 29
  - 7.2 Application Flowchart 30
  - 7.3 Java™ Classes 31
    - 7.3.1 KioskEngine Class 31
    - 7.3.2 ParseXML class 36
    - 7.3.3 StringJEditorPane class 39
    - 7.3.4 ImageJLabel class 41
    - 7.3.5 AudioCombo class 43
    - 7.3.6 VideoCombo class 44
    - 7.3.7 ScreenString class 46
    - 7.3.8 ScreenImage class 47
    - 7.3.9 ScreenTextArea class 48
    - 7.3.10 ScreenAudio class 49
    - 7.3.11 ScreenVideo class 50

8. Proof of Concept 52
  - 8.1 Useful Hints for putting together a Kiosk using KioskEngine 52
  - 8.2 Pilot Application : Columbus State University (CSU) Kiosk 53

9. Extensibility of KioskEngine 56

10. Conclusion 57

References 58

Appendix: XML file for Columbus State University Kiosk 59

List of Figures.

Figure 1. Super Bowl (1998), at San Diego, CA 5

Figure 2. New Line Cinema (Lost In Space), at various countries 5

Figure 3. Screenshot of a Kiosk-in-a-Box 6

Figure 4. Ticketing Kiosk 7

Figure 5. Information Kiosk 8

Figure 6. Conceptual Class Diagram for Classes of Kiosk Engine. 29

Figure 7. Flowchart describing the Operation of Kiosk Engine. 30

Figure 8. Class Diagram of the KioskEngine Class 31

Figure 9. Class Diagram of the ParseXML Class 36

Figure 10. Class Diagram of the StringJEditorPane Class 39

Figure 11. Class Diagram of the ImageJLabel Class 41

Figure 12. Class Diagram of the AudioCombo Class 43

Figure 13. Class Diagram of the VideoCombo Class 44

Figure 14. Class Diagram of the ScreenString Class 46

Figure 15. Class Diagram of the ScreenImage Class 47

Figure 16. Class Diagram of the ScreenTextArea Class 48

Figure 17. Class Diagram of the ScreenAudio Class 49

Figure 18. Class Diagram of the ScreenVideo Class 50

Figure 20. Snapshot of Start Screen of the CSU Kiosk 53

Figure 21. Snapshot of Tower Screen of the CSU Kiosk 54

List of Tables.

Table 1. Comparison of Features of Kiosk Products. 10

Table 2. XML tags in XMLKiosk file for KioskEngine 12

Table 3. Comparison of Programming Languages 18

Acknowledgements

It is a fortune to find opportunities for thanking those people towards whom one feels a deep sense of gratitude and respect.

I am especially indebted and very thankful to *Dr. Stan Kurkovsky*, my thesis advisor and Assistant Professor, Department of Computer Science, who delegated me with the responsibility of developing such a wonderful project and with his confidence he provided me the opportunity of working with him and his valuable guidance throughout.

I am honored to extend my gratitude and respect to my parents who have been a source of encouragement for me.

I am also greatly thankful to the following faculty members for their valuable encouragement and support,

Dr. Wayne Summers, Professor and Distinguished Chairperson, Department of Computer Science

Dr. Bhagyavati, Assistant Professor, Department of Computer Science

Dr. Vladimir Zanev, Associate Professor, Department of Computer Science

With Regards,

Manish S. Shah

1. Introduction

Conveying a message in a best way requires high quality communication being done with easy-to-access information. Graphical information supplemented with audio/video information furnishes the best quality of message conveyed. Nowadays it is the focus of every entrepreneur to save a customer's time and minimize manual intervention wherever possible in order to save capital and maximize profit. Best suited in such environments are Kiosks that provide all the required information to customers at virtually any location with minimum monitoring efforts.

Literally, Kiosk is an interactive terminal with a display device, which presents various types of information on a screen. Kiosks are typically used in locations where self-service is required or something is to be displayed just for the sake of providing information, for e.g. theatres, banks, convention centers, visitor stations, etc. They can include various types of digital multimedia information such as images, decorative text, audio, video, etc.

Retail kiosks are being incorporated by a large variety of retailers since first introduced in 1980s [Douglas00] in wide business areas such as banking, gaming, internet access, self-checkout terminals, providing service/product information, etc. to name a few. They have changed the paradigm of presenting multimedia information to the target viewers. Needless to say, not far ahead in future, Kiosks are going to redefine the means of electronic communication and the ways to do business.

2. Motivation

Transparent to the end user, the task of developing a kiosk is fairly time consuming and nothing but easier said than done. Evidently, there is only a small number of companies in the industry totally focused on development of kiosks and very few of them developing with some sort of generalized software for all kiosks. Most of the kiosks once developed are good for targeted systems only and do not provide any options for an easy change in configuration.

Scarcely realized fact is that kiosks, like other software products, need two basic things to keep it working, viz. data and processing power, but it is possible to draw a line between the two. Isolating the two enables us to focus only on the data (images, audio, video, etc.) and their organization and relieves us from worrying about developing the processing power again and again. Focus of this thesis is to develop such a generic program which acts as a processing power behind virtually any kiosk which intends to use wide range of supported types of multimedia data.

More importantly,

What changes from one kiosk to another is the multimedia data and their specified organization.

Thereby, it is our intent to research and develop an easy and fast way of generating multimedia kiosks and backing them up with a generalized processing power as a light weight application which can occupy very little memory and hence can run on a wide range of systems including thin clients such as PDA's or wireless devices. Such a concept will relieve lot of companies from programming for each kiosk on demand; instead they would be able to speed up the process by focusing just on the organization of their multimedia data, which is of paramount importance for more informative and appealing kiosk.

2.1 Desired Features

Some of the features in mind before starting on the actual development of Kiosk Engine were that the driving program should be able to present information differently from one kiosk to another, based on the different inputs provided. The means of providing input data should be cost-effective (meaning that it should not require expensive software or excessive time) and easy to create, modify and store. Also, the program should be a lightweight application and be able to run on majority of the platforms out in the market. Besides, the five basic types of multimedia data to be supported by the resulting kiosks driven by Kiosk Engine are text, images, documents (containing plain/rich text or HTML), audio and video.

Taking into account these basic facts it is essential to explore, what do the companies in the current Kiosk market has to offer, and then evaluate the chances of existing products filling the gap of our needs and/or thus substantiate the probability of success of Kiosk Engine to be able to provide adequate usefulness and fruitful results to it's users.

3. Survey of Other Products on the Market

3.1 The Apunix Kiosk Engine for the Java™ Platform

Apunix [Apunix03], a software development company based in San Diego, CA, has developed a multi-functional solution for kiosk development based on Java. With goals of developing a robust, reliable and scalable solution for kiosks, Apunix have created a software that provides a content creation tool and kiosk engine using which customer can create a custom solution to meet their own needs and run it using their kiosk engine. As promised by Apunix, they developed a platform independent solution in Java™ programming language, which integrates the multimedia data with the engine to create a graphically rich kiosk.

The user has to use their development tool in order to specify the multimedia data and design the kiosk, which at times may turn out to be time consuming and not so user-friendly. Moreover, the logic of the flow of screens in the kiosk is also to be developed using the tool, which is not the easiest possible solution. However, the kiosks developed can be graphically rich containing text, images, audio, video, etc. and attractive depending upon the creativity of the developer.

Some of the kiosks developed by Apunix using their product were deployed successfully at several locations, the screenshots of some of which are shown next.

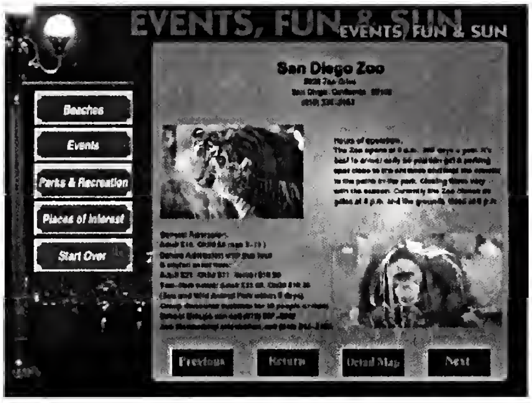

EVENTS, FUN & SUN
EVENTS, FUN & SUN
San Diego Zoo
Beaches
Events
Parks & Recreation
Places of Interest
Start Over
Previous
Return
Detail Map
Next

Figure 1. Super Bowl (1998), at San Diego, CA

THE TRAILER
THE MAKING OF
THE CAST

Figure 2. New Line Cinema (Lost In Space), at various countries

This product is possibly the closest match to our aims, however, since it is not a light weight application and it requires the development tool to be used to provide the input, it does not exactly meet the exact feature requirements desired, as will be summarized later.

3.2 Kiosk-In-A-Box from Mass Multimedia, Inc

Mass Multimedia Inc. [MassMlt03], a Colorado based company that offers touch-screen systems since 1995, have developed a software called Kiosk-In-A-Box for developing touch screen kiosk applications without programming. It is a software application development tool that allows a user to create multimedia-based kiosks for Internet/local deployment without any prior knowledge of programming. The product does however suggest the user to have a basic knowledge on how to generate computer graphics. The kiosks developed can contain images, text, sound and video clips and is capable of running on Windows operating systems only.

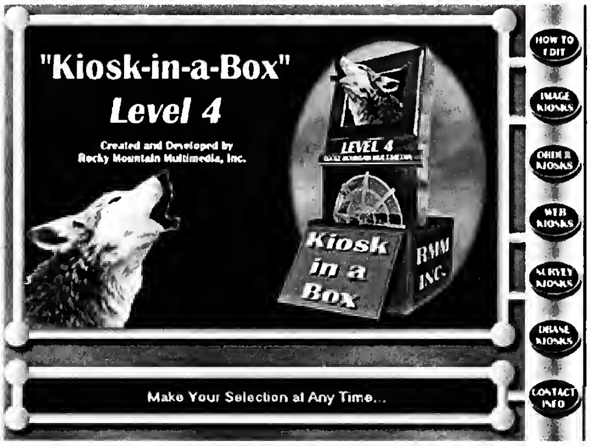

"Kiosk-in-a-Box"
Level 4
Created and Developed by
Rocky Mountain Multimedia, Inc.
LEVEL 4
Kiosk in a Box
RMM INC.
HOW TO EDIT
IMAGE KIOSKS
OTHER KIOSKS
WEB KIOSKS
SURVEY KIOSKS
DBASE KIOSKS
CONTACT INFO
Make Your Selection at Any Time...

Figure 3. Screenshot of a Kiosk-in-a-Box

3.3 KioskEngine™ from Touch Controls, Inc.

Touch Controls Inc. [Touch03], a California based company, has developed KioskEngine™, which is a web-based software engine that allows a user to create a custom kiosk multimedia program with a pre-structured built-in template. It provides pull down menus and pre-configured graphic elements using which, a user, without any programming knowledge can add text, photos, graphics and video clips and develop intuitive web-based kiosk. KioskEngine™ can be hosted using either MS Personal Web Server [MS03] or MS Internet Information Server [MS03] and is displayed and administered using MS Internet Explorer® 4.0 [MS03] or higher web browser. Consequently, the kiosks powered by KioskEnigne™ can be used only on Windows 95/98 or NT operating systems.

Following are a couple of screen-shots of point-of-sale kiosks provided by this company.

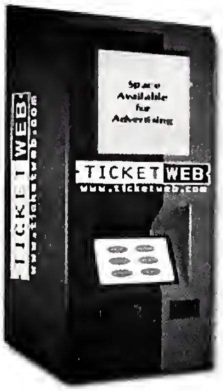

Space Available for Advertising
TICKET WEB
www.ticketweb.com
TICKET WEB
www.ticketweb.com

Figure 4. Ticketing Kiosk

Figure 5. Information Kiosk

3.4 Our Product: Kiosk Engine (KE)

The program developed by us is called Kiosk Engine (KE), which can work as a driver of a kiosk when provided with the input consisting of multimedia data, their description and organization on the kiosk. Apart from the supported functionalities to be discussed in brief below, there are a few additional functionalities of the Kiosk Engine that are worthwhile to be noted here. The input to the program is an XML file, which describes the sources, organization and presentation order and style of multimedia data. Kiosk Engine parses the XML file, extracts the information and stores them in different types of objects for each tag in the XML file, the format of which is to be described later.

Unlike most of the kiosks, which are platform dependent, Kiosk Engine was aimed to be *platform independent* and henceforth run on virtually any platform with basic media framework support consisting of JMF libraries [SunJMF03]. To achieve platform independence, KE is programmed in the latest version of Java™ 2 programming language, J2SE™ SDK v 1.4.0_01 [SunJ2SE03], which is available as of now. Besides, to allow the use of multimedia data, Sun Microsystems, Inc., provides with a framework/API called Java Media Framework [SunJMF03] to be used with Java programs that intend to provide support for wide range of audio/video formats. It's fair to stipulate henceforth, that Kiosk Engine can run either in a web browser as a Java™ applet or standalone in a Java™ frame, on any Java Virtual Machine [LindholmY03] that uses Java™ Runtime Environment, J2SE™ JRE v 1.4.0_01 [SunJ2SE03] and has JMF libraries installed.

A brief discussion of input XML file, which describes the multimedia data to be supported by Kiosk Engine, follows shortly. First, it will be informative to briefly summarize the comparison of our Kiosk Engine with other products previously mentioned.

3.5 Comparison of Products

Table 1 lists features against products and compares what our Kiosk Engine has to offer compared to other products.

Table 1. Comparison of Features of Kiosk Products.

	Our Kiosk Engine	Apunix	Mass Multimedia	Touch Controls
Text support	Yes	Yes	Yes	Yes
Images support	Yes	Yes	Yes	Yes
HTML document (.htm) support	Yes	Yes	Yes	Yes
Audio files support	Yes	Yes	Yes	Yes
Video files support	Yes	Yes	Yes	Yes
Template-free design (i.e. no predefined template designs to be used only)	Yes	Yes	N/A	No
Easy text inputs	Yes	No	No	No
Web-based or local kiosk	Yes	Yes	Yes	Web-based only
Light weight application	Yes	No	No	No
Platform Independent	Yes	Yes	No	No
Software requirements	JVM, JMF	JVM, Database support	Win 95 or higher	IIS or PWS, IE 4.0 or higher, Win 95/98/NT

4. Data Representation in Kiosk Engine

The format of the input file was chosen to be of XML type for two favorable reasons. The first reason, we needed to be able to create new tags for each basic type of data as an on-demand basis and specify custom attributes for each tag. The second reason being, that it should be easy to parse the file using Java™ to avoid creating a custom parser which would add more classes to the code and processing time to the end product. XML (eXtensible Markup Language) allows us to create new tags and specify custom attributes to each tag. Java™ on the other hand has built in parser classes for viewing the XML file as both, a Document using DOM (Document Object Model) classes or on an event-driven basis using SAX (Simple API for XML) classes.

4.1 The Structure of XML file for KE

This section describes the semantic structure of the XML file for KE, which we call as XMLKiosk. For each Kiosk application to be developed there is a root node <Kiosk> in the XMLKiosk file, which contains all the different tags to describe the entire kiosk. In XML description tags are also referred as nodes or elements and each tag has a starting and ending tag. At the next depth level there are two elements called <header> and <screen>, which are the containers of different types of multimedia objects and both of which can have all or any of the tags for the multimedia objects. In the context of this project, the input XML file is designed to describe five basic types of multimedia data, viz. text, images, documents, audio and video; each of which has tags called <text>, <img>, <textArea>, <aud> and <vdo> respectively. For each instance of any of the data types on the kiosk there is a corresponding tag describing it in the file. The <text> and <img> tags occasionally have <action> tag(s) to facilitate the browsing of kiosk from one screen to another. Also each <aud> or <vdo> tag can have multiple <file> tags which allows a single audio/video instance to allow playing multiple files alternatively.

Below is a summary of the different types of tags in a typical XMLKiosk file:

Table 2. XML tags in XMLKiosk file for KioskEngine

Element/Tag Name	**Parent Tag**	**Attributes**	**Comments**
Kiosk	None	title, red, green, blue, start	Root element of the multimedia kiosk XML descriptor. Start specifies the id of the start screen for the kiosk.
Header	kiosk	Id	Header element has same contents as a Screen element, but it is used by screens to display common objects among different screens.
Screen	kiosk	id, header	Screen is the main element of the kiosk. Each screen has a id using which it is identified by the interpreter. The header attribute has the id of the header used by the screen.
Text	header / screen	type, font, style, size, xpos, ypos, width, height, color	Represents a small text element, the text displayed will be of a uniform format.
Img	header / screen	src, alt, xpos, ypos, width, height	Represents an Image to be displayed on the kisok. The type can be JPEG or GIF.
Action	text/img	Type [alt, dxpos, dypos, dwidth, dheight]	Defines when the action is to be performed (ex: onclick)
Target	action	None	Represents the target of the action, ie. typically an id of an element / tag on/of the screen.
Textarea	header / screen	src, type, xpos, ypos, width, height	Represents a document to be displayed from the specified relative source. The document can be of type .doc or .htm or plain text. HTML 3.0 version supported.
Aud	header / screen	start, xpos, ypos, width, height	Represents an instance of audio player to be displayed.
Vdo	header / screen	start, xpos, ypos, width, height	Represents an instance of video player to be displayed.
File	aud/vdo	id, title, src	Represents each audio/video file in a player.

5. Implementation of Kiosk Engine

5.1 Other languages and development tools

Recalling the desired features that are discussed before, let us try to evaluate the choices of different programming languages available, and justify that Java™ is the best suited programming language to our needs in the present context.

Although there are a number of different programming languages and development tools available, it is far beyond the scope of this report to list all of them. Thus, we will only list a couple of languages/development tools that has the best chances to be useful in fulfilling our requirements.

5.1.1 Microsoft® Visual Basic 6 and Visual Basic .NET

MS Visual Basic® 6 [MS03] is an event-driven Rapid Application Development (RAD) tool which comes with a number of built-in ActiveX® [MSDN03] controls and reference libraries to support development of multi-functional commercial applications. Playing selected formats of audio and video is possible using *Multimedia control* [MSDN03]. The functions provided by the *Multimedia control* depend on the hardware and software configurations of the machine as it relies partly on the driver support that comes with the Windows® operating system. Popular file formats supported by *Multimedia control* are .mid, .wav and .avi. Besides, there are other controls such as *PictureBox* and *Label* for displaying images and text, respectively. For working with XML, there comes a library containing MSXML parser that exposes functions to work with XML files.

MS Visual Basic.NET [MS03] is the latest version of Visual Basic that comes with similar controls and libraries such as *Microsoft Windows Media® Player 9 Series ActiveX control* for audio/video and MSXML parser to work with XML.

Although, Visual Basic 6 and its .NET version has enough support to satisfy our requirements in programming aspect, but, it does accompanies a few prices to be paid in on the functional aspects desired for our project. An application developed in Visual Basic has to be bundled with referenced library files for proper functioning, which will take up more memory space on the target machine. Also, the application has to be installed properly on the target machine which intends to run it, which predominantly, can be only those machines that have Windows® 95 or higher operating system.

As it is evident from above, Visual Basic does not meet our requirements in totality and hence it is not much feasible to develop Kiosk Engine using Visual Basic.

5.1.2 Macromedia® Flash

Macromedia Flash [MacroFlash03] is a development tool for creating rich Internet content. It provides the users with tools to embed audio, video and other graphically rich digital media objects into their application (a Flash movie file with .swf extension) and create attractive and visually appealing front-ends.

Playing audio is easy as audio files can be imported by selecting the location and can be embedded in the Flash movie. Flash MX, the latest version, allows embedding video files of formats such as MOV (QuickTime), AVI (Audio Video Interleave) and MPG/ MPEG (Motion Picture Experts Group) into a Flash movie [MacroVid03]. Flash uses Sorenson Spark video codec to compress embedded video and store them in a high quality but smaller file size. Tools for adding text and images also exist in Flash.

Flash is a streaming technology, which means that it is based on time frames and hence the content displayed in a Flash movie is dependent on the Time Frame in which it is added. Besides, Flash movies require Flash Player for displaying it. Since audio and video files are embedded in the movie file, it increases the size of the movie and also contradicts to our idea of separating data completely from the processing. Moreover, working with XML is not supported natively in Flash and adding multimedia content in the movie dynamically, is not possible. Adding more to it, since data is embedded within the movie (.swf) file, it cannot be changed without reformatting the movie content. Hence, it is clear that Flash does not fit into our requirements and cannot offer what we desire to implement using it.

5.2 Sun® Java™

Java™ is an object-oriented programming language developed by Sun Microsystems®, Inc., to develop standalone applications or web-based applets. For writing programs using Java™, Sun Microsystems® provides development toolkit called Java Development Toolkit (JDK) which comes with numerous built in application programming interfaces (APIs). These APIs provide number of interfaces and classes which can be imported in programs and used to work in wide variety of programming areas. Java™ is platform independent, which means that programs written in Java™ can work on all major platforms without being compiled again. The fact that makes this possible is the JVM [LindholmY03] or in other words the Java™ Runtime Environment (JRE) been installed on the target machine. The programs written in Java™, when compiled, gets converted to bytecode which can be then interpreted by any Java Virtual Machine (JVM). A Java Virtual Machine is any machine that has JRE [SunJ2SE03] installed on it. Java programs work on a wide range on platforms including Windows™, Solaris™, Linux and virtually on any computing hardware including Thin Clients, PDA's, embedded devices, workstations and servers.

Swing is a part of Java Foundation Classes (JFC) [SunSwing03], which has built-in features to help us develop graphical user interfaces (GUIs) and provides readymade "lightweight" components to be used in applications/applets. Swing provides pluggable look and feel and thus same program using Swing components can use Java™ look & feel or the system's native look and feel. The components that are used for displaying images and text in Kiosk Engine are called *ImageIcon* and *JEditorPane*, respectively. Additionally, there is a *JLabel* component used to wrap around the *ImageIcon* and a *JScrollPane* component that wraps around *JEditorPane* component to give it the scrollable functionality. Moreover, as mentioned earlier, we also need support for displaying documents, which is possible by using the same *JEditorPane* component. *JEditorPane* is a text component whose behavior is dependent upon the type of the document it is given to display. Depending on the type of the content, *JEditorPane* uses the appropriate *EditorKit* to load the content and effectively morphs into the proper kind of text editor.

Playing audio and video files from Java™ programs is easy with Java™ Media Framework (JMF) API [SunJMF03]. JMF enables Java applications and applets to work with audio, video and other time-based media. JMF API is an optional package that extends the multimedia capabilities of J2SE™ and helps developers in making scalable multimedia programs that can capture, playback and stream multiple media formats. With a range of encoders and decoders provided by JMF API, it can offer support for various rich media formats such as AVI and MPEG for video and MP3 for audio. Moreover, it provides interfaces and classes, which can return visual components for controlling the audio and video files at runtime.

Java programs can be written for standalone applications as well as web-based applets. Standalone applications are those applications that are intended to run on a single machine at one time. Applets however, are run from a Java enabled World Wide Web browser. Applets are embedded into web pages, and a reference to the main class is stored in the web page. While displaying the web page, the Java enabled browser uses the reference to instantiate the applet on the local machine using the built in Java interpreter.

Java provides APIs to work with XML in both contexts; Document Object Model (DOM) and Simple API for XML (SAX). The packages in J2SE that has the built in parsers for DOM and SAX are *org.w3c.dom*, *org.xml.sax* and *javax.xml.parsers*. The DOM views an XML file as a document and generates object instances for each tag inside the document, whereas the SAX is an event-based parser that parses the XML file one object at a time as and when an event triggers the action of parsing that tag.

For the context of our Kiosk Engine, we need to make the kiosk to be accessible both standalone as well as web-based. To realize this idea, keeping in mind that lightweight program was what we originally wanted to achieve, we use the same program for both standalone as well as web-based purposes. For this, we make use of the main applet for web-based access and the same applet embedded in a Java Frame for standalone purposes.

5.3 Comparison of Java™ with other languages and tools

A summary of comparison of Java & other languages/tools discussed above, is provided in form of features vs. languages/tools, in the table below:

Table 3. Comparison of Programming Languages

	Java™	Visual Basic & Visual Basic .NET	Flash
Text, HTML	Yes	Yes	Yes
Image	Yes	Yes	Yes
Audio, Video	Yes	Yes	Yes
Data separate from program	Yes	Yes	No
Standalone and web-based	both	Standalone	Web-based
Platform independent	Yes	No	No

It is clear from the previous discussion that Java meets our desired requirements, offers all of the favorable API support that we can make use of in programming and it is certainly the best fit for developing our Kiosk Engine (KE).

6. Content Requirements

6.1 Elements of XMLKiosk file for KE

Previously, we discussed the structure of the XMLKiosk file for KE in brief. This section discusses the XMLKiosk file in detail with a detailed explanation of each element/tag/node in the XMLKiosk file and its attributes, how they are used in the program and what purpose they serve on the Kiosk screen. Also we will discuss the depth of each tag (assuming depth starts at zero) and the constraints associated with it, such as optional or maximum number of such tags allowed.

The Parser used for parsing the XML file is based on Document Object Model (DOM) in terms of which an XML file is referred to as a document.

6.1.1 <kiosk> tag

Example of the tag in an XMLKiosk file:

```
<?xml version = "1.0" ?>
<kiosk red="0" green="0" blue="100" start="screen1">
        <header id="header1">
                ...
        </header>
        <screen id="screen1" header="header1">
                ...
        </screen>
</kiosk>
```

This is the root node of the XML document, so depth is 0. For each Kiosk application to be made there is only one <kiosk> tag in the XML file. Note, that the spelling and the format of all the tags should be exactly same as that specified in the example. All other tags are inside this tag, but the direct children are <header> and <screen> tags.

Attributes

- *red*, *green*, *blue* : specify the background color of the kiosk when converted by the program into RGB value.
- *start* : specifies the id of the first screen node to be processed and displayed in this kiosk. This id value has to be the same as the unique id value specified in the corresponding screen tag.

6.1.2 <screen> tag

Example of the tag in an XMLKiosk file:

```
<?xml version = "1.0" ?>
<kiosk red="0" green="0" blue="100" start="screen1">
        <header id="header1">
                ...
        </header>
        <screen id="screen1" header="header1">
                <aud> ... </aud>
                <vdo> ... </vdo>
                <img> ... </img>
                <text> ... </text>
                <textArea> ... </textArea>
        </screen>
</kiosk>
```

This is an optional tag and the depth is 1 as it is the direct child of root node. For each Kiosk application to be made there can be upto 100 <screen> tags in the XML file. There are five optional tags that can be contained in this tag corresponding to audio, video, image, text and text documents to be displayed in one screen.

Attributes

- *id* : serves as the identification of this screen for <action> tags in the kiosk. Each <screen> tag in the kiosk should have a unique id value (different from id values of other screens in the kiosk).
- *header* : specifies the id of the header node to be processed and displayed along with this screen. At runtime, the value of *header* attribute is used only if it is different than the current header displayed.

6.1.3 <header> tag

Example of the tag in an XMLKiosk file:

```
<?xml version = "1.0" ?>
<kiosk red="0" green="0" blue="100" start="screen1">
        <header id="header1">
                <aud> ... </aud>
                <vdo> ... </vdo>
                <img> ... </img>
                <text> ... </text>
                <textArea> ... </textArea>
        </header>
        <screen id="screen1" header="header1">
                ...
        </screen>
</kiosk>
```

This is an optional tag and the depth is 1 as it is the direct child of root node. For each Kiosk application to be made there can be any number of <header> tags in the XML file. There are five optional tags that can be contained in this tag corresponding to audio, video, text and text documents to be displayed in one screen.

Header nodes serves the purpose of displaying common objects between different screens, which helps in minimizing the duplication of the same tag in multiple screen tags. At runtime, the header displayed remains unchanged until a new header is referenced. Multiple screens can share the same header.

Attributes

- *id* : serves as the identification of this header for the <screen> tags in the kiosk. Each <header> tag in the kiosk should have a unique id value.

6.1.4 <text> tag

Example of the tag in an XMLKiosk file:

```
<text  font="Serif"  style="BOLDITALIC"  size="16"  xpos="200"
ypos="55" width="200" height="30" color="LIGHT_GRAY">
        Text to be displayed goes here
        <action ... >   <target> ... </target>
        </action>
</text>
```

This is an optional tag and the depth is 2. For each <screen> & <header> pair of tags combined there can be upto 100 <text> tags (At any time there is only one screen/header pair being displayed on the kiosk screen). This restriction is placed in the program for performance issues and it can be changed in the program. The text to be displayed appears between <text> and </text> tags. It can optionally contain one <action>/<target> pair of tags to facilitate browsing from one screen to another.

Attributes

- *font*, *style*, *size, color* : specify the appearance of the text on screen.

- *xpos*, *ypos*, *width*, *height* : Specifies the size and location of the text.

Possible values of attributes

- font : Serif, SansSerif, Monospaced, Dialog, DialogInput.
- style: BOLD, ITALIC, BOLDITALIC. All other values will result in PLAIN font.
- color: BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW. All other values will result in default BLACK color.

6.1.5 <img> tag

Example of the tag in an XMLKiosk file:

```
<img src="foldername/filename.jpg" alt="About this Image"
xpos="575" ypos="80" width="120" height="465">
        <action ... >
                <target> ... </target>
        </action>
</img>
```

This is an optional tag and the depth is 2. For each <screen> & <header> pair of tags combined there can be upto 100 <img> tags (At any time there is only one screen/header pair being displayed on the kiosk screen). This restriction is placed in the program for performance issues and it can be changed in the program. Each <img> tag represents an image to be displayed on screen. It can optionally contain upto 50 <action>/<target> pair of tags to facilitate browsing from one screen to another.

Attributes

- *src* : specifies the relative location of the image file.
- *alt* : specifies a very short description or title for the image. It is displayed on screen during some action, for e.g. when mouse cursor moves on the image.
- *xpos*, *ypos*, *width*, *height* : Specifies the size and location of the image.

6.1.6 <action> and <target> tags

Example of the tag in an XMLKiosk file:

```
<action type="onclick" alt="About this rectangle part of image"
dxpos="10" dypos="10" dwidth="92" dheight="70">
            <target>
                        screen1
            </target>
</action>
```

Both tags are optional but if they appear both have to be present to make sense. The depth of <action> tag is 3 (its parent being a <text> or <img> tag) and that of <target> tag is 4 (<action> tag being the parent). It facilitates browsing from one screen to another. The value between the <target> tags specifies the id of the screen to be displayed as a result of this action.

Attributes

- *type* : specifies when the action is to be performed (for e.g. “onClick”).
- *[alt]* : optional attribute only for <action> tag, specifies a very short description or title for this rectangular part of the image. It is displayed on screen during some

action (for e.g. when mouse cursor moves on the image) on this rectangular part of image.

- *[xpos, ypos, width, height]* : optional attributes only for <action> tag, specifies the size and location of the rectangular part of image.

6.1.7 <textArea> tag

Example of the tag in an XMLKiosk file:

```
<textArea src="foldername/filename.htm" xpos="75" ypos="80"
width="120" height="465">
</textArea>
```

This is an optional tag and the depth is 2. For each <screen> & <header> pair of tags combined there can be upto 100 <textArea> tags (At any time there is only one screen/header pair being displayed on the kiosk screen). This restriction is placed in the program for performance issues and it can be changed in the program. Each <textArea> tag represents a document (.htm/.doc/.txt) from a file to be displayed on screen.

Attributes

- *src* : specifies the relative location of the document file.
- *xpos, ypos, width, height* : specifies the size and location (rectangular area) where the document is to be displayed on screen.

6.1.8 <aud> tag

Example of the tag in an XMLKiosk file:

```
<aud    start="song2"    xpos="200"    ypos="95"    width="350"
height="25">
        <file ... ></file>
</aud>
```

This is an optional tag and the depth is 2. For each <screen> & <header> pair of tags combined there can be upto 10 <aud> tags (At any time there is only one screen/header pair being displayed on the kiosk screen). This restriction is placed in the program for performance issues and it can be changed in the program. Each <aud> tag represents an audio player to be displayed on screen and can contain upto 25 <file> tags inside it.

<u>Attributes</u>

- *start* : specifies the id of the audio file to be started by default. The value should correspond to the one of the id values of the <file> tags inside.
- *xpos*, *ypos*, *width*, *height* : specifies the size and location where the audio player is to be displayed on screen.

6.1.9 <vdo> tag

Example of the tag in an XMLKiosk file:

```
<vdo    start="song2"    xpos="200"    ypos="95"    width="350"
height="25">
        <file ... ></file>
</vdo>
```

This is an optional tag and the depth is 2. For each <screen> & <header> pair of tags combined there can be upto 10 <vdo> tags (At any time there is only one screen/header pair being displayed on the kiosk screen). This restriction is placed in the program for performance issues and it can be changed in the program. Each <vdo> tag represents a video player to be displayed on screen and can contain upto 25 <file> tags inside it.

Attributes

- *start* : specifies the id of the video file to be started by default. The value should correspond to the one of the id values of the <file> tags inside.
- *xpos*, *ypos*, *width*, *height* : specifies the size and location where the video player is to be displayed on screen.

6.1.10 <file> tag

Example of the tag in an XMLKiosk file:

```
<vdo start="song1" xpos="200" ypos="95" width="350" height="25">
        <file   id="song1"   title="File   title   on   screen"
                 src="foldername/filename.mp3">
        </file>
</vdo>
```

This is an optional tag and the depth is 3. For each <aud> or <vdo> tag there can be upto 25 <file> tags. This restriction is placed in the program for performance issues and it can be changed in the program. Each <file> tag represents an audio/video file to be played in the corresponding audio/video player on screen.

Attributes

- *id* : specifies the id of this file in this player. The value is used by the parent <aud>/<vdo> tag to reference this file.
- *title* : specifies a short title for this file to be displayed in the player on the screen.
- *src* : specifies the physical location of this file.

7. Program Architecture of Kiosk Engine

This section explains the overall architecture including significant details of each class of the Kiosk Engine application. It starts with the conceptual class diagram followed by the application flowchart and then explanation of each individual class and its significance in the application.

7.1 Conceptual Classes of Kiosk Engine

Figure 6 presents a diagram of conceptual classes used in our implementation of Kiosk Engine.

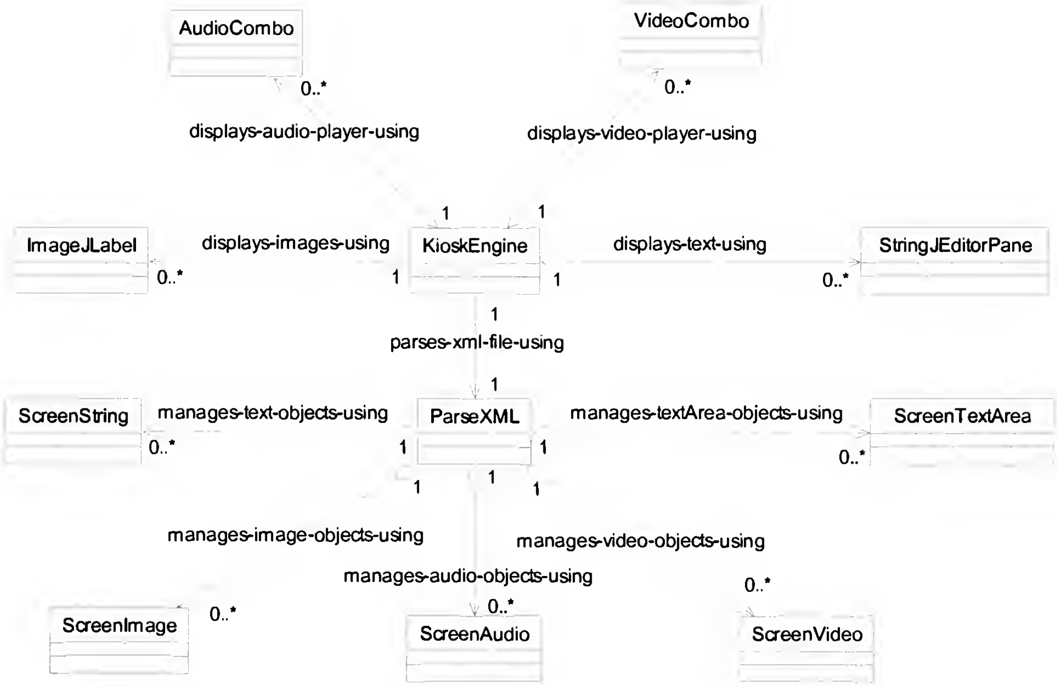

AudioCombo
VideoCombo
0..*
0..*
displays-audio-player-using
displays-video-player-using
1
1
ImageJLabel
displays-images-using
KioskEngine
displays-text-using
StringJEditorPane
0..*
1
1
0..*
1
parses-xml-file-using
1
ScreenString
manages-text-objects-using
ParseXML
manages-textArea-objects-using
ScreenTextArea
0..*
1
1
0..*
1
1
1
manages-image-objects-using
manages-video-objects-using
manages-audio-objects-using
0..*
ScreenImage
0..*
0..*
ScreenAudio
ScreenVideo

Figure 6. Conceptual Class Diagram for Classes of Kiosk Engine.

7.2 Application Flowchart

Figure 7 presents a flowchart describing the operation of Kiosk Engine.

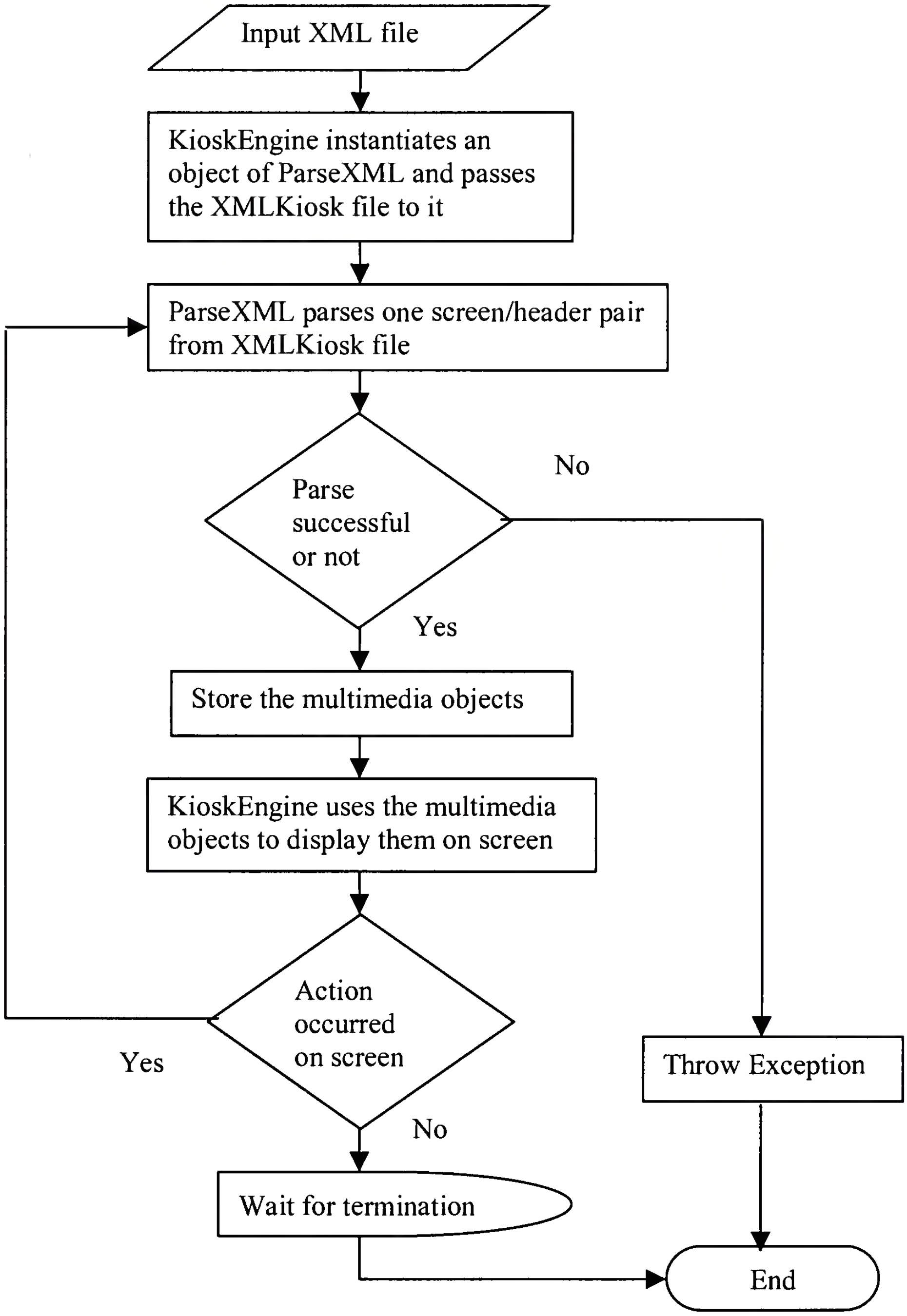

Input XML file
KioskEngine instantiates an object of ParseXML and passes the XMLKiosk file to it
ParseXML parses one screen/header pair from XMLKiosk file
Parse successful or not
No
Yes
Store the multimedia objects
KioskEngine uses the multimedia objects to display them on screen
Action occurred on screen
Yes
No
Throw Exception
Wait for termination
End

Figure 7. Flowchart describing the Operation of Kiosk Engine.

7.3 Java™ Classes

Altogether there are eleven Java classes in Kiosk Engine application. This section examines the details of each class in terms of its dependence on and usefulness for other classes in the application and/or the role it plays for the successful execution of application. Each subsection discusses one class in detail describing some of the important methods in the class supplemented with pseudo code wherever significant.

7.3.1 KioskEngine Class

Figure 8 presents the class diagram of the KioskEngine Class.

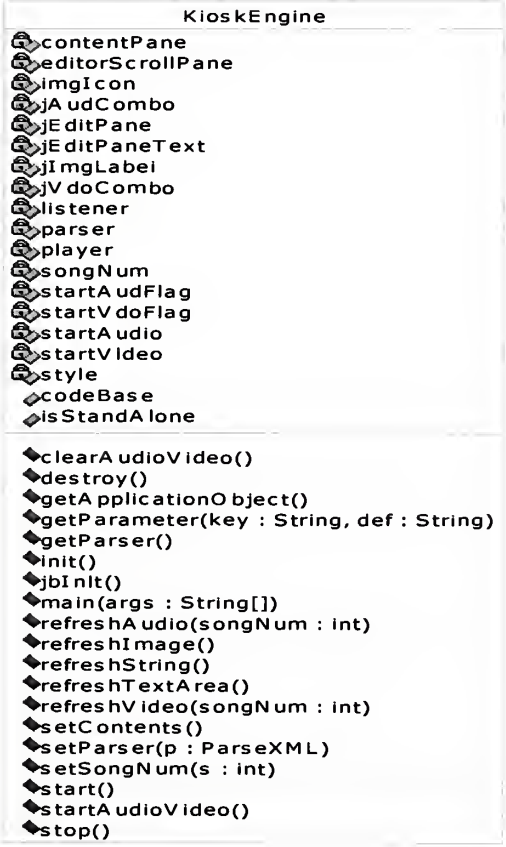

KioskEngine
contentPane
editorScrollPane
imgIcon
jAudCombo
jEditPane
jEditPaneText
jImgLabel
jVdoCombo
listener
parser
player
songNum
startAudFlag
startVdoFlag
startAudio
startVideo
style
codeBase
isStandAlone
clearAudioVideo()
destroy()
getApplicationObject()
getParameter(key : String, def : String)
getParser()
init()
jbInit()
main(args : String[])
refreshAudio(songNum : int)
refreshImage()
refreshString()
refreshTextArea()
refreshVideo(songNum : int)
setContents()
setParser(p : ParseXML)
setSongNum(s : int)
start()
startAudioVideo()
stop()

Figure 8. Class Diagram of the KioskEngine Class

This class extends directly from *javax.swing.JApplet*. Consequently, KioskEngine is an Applet and hence it can run either in a web browser embedded within a HTML page or standalone embedded in a Java Frame (an instance of class *javax.swing.JFrame*). It is the main class (or the driver class) in the application and it is referenced directly by the HTML page or Java Frame in which the kiosk runs. The important packages imported by this class are *javax.swing* and *javax.media*.

When referenced from a HTML page, the path of the input XML file should be passed as a parameter to this class relative to the directory in which this class exists. If referenced from a Java Frame the input XML file should be named *"kiosk.xml"* and placed in a folder named *"xml"* which resides in the directory in which this class exists. This restriction in naming is placed in order for the application to be able to execute with the click of a mouse button instead of a command prompt.

Following is a pseudo code for this class:

1. Create an object of ParseXML and instantiate it with the path of XML file.
2. Parse the XML file using the instance of ParseXML.
3. Use the objects corresponding to the screen/header pair parsed by the instance of ParseXML and use them to display the multimedia objects on the screen.
4. On termination, destroy any non-null objects remaining.

The execution order of the methods of this class is different at different times and depends on the mode (standalone or from web browser) it is been executed in. It does have a *main* method which is used only when the class is supposed to run as a standalone application. The purpose of the *main* method is to create an instance of Java Frame and embed KioskEngine applet inside it. This is required as an Applet cannot run standalone by itself. When referenced from within a HTML page, the browser calls the methods of this class in a predefined order being *init(), start(), stop()* and *destroy()*. One of the important methods which is called by *start()* method and which is responsible for the addition of multimedia objects on the applet is *jbInit()*. This method calls the individual methods for displaying audio, video, text, image and documents in that order. Each of those methods uses the objects in the parser to instantiate each object's position, size and appearance on the screen.

Following are the methods corresponding to each multimedia object:

- *refreshAudio()*: Corresponds to the <aud> tags in the XML file. Used for display of audio players on the screen using instances of AudioCombo class. It populates the combo box of each audio player with the sound files corresponding to it and associates a control panel component to each player which facilitates forward, rewind, start, stop and pause functions. It also allows controlling the volume and the rate at which the file is to be played. Most popular audio file types supported are .wav, .mid, .mp3 and .au.
- *refreshVideo()*: Corresponds to the <vdo> tags in the XML file. Used for display of video players on the screen using instances of VideoCombo class. It populates the combo box of each video player with the video files corresponding to it and associates a control panel (similar to the audio player) and a visual component to

display the video into. Most popular video file types supported are .avi, .mov and .mpeg.

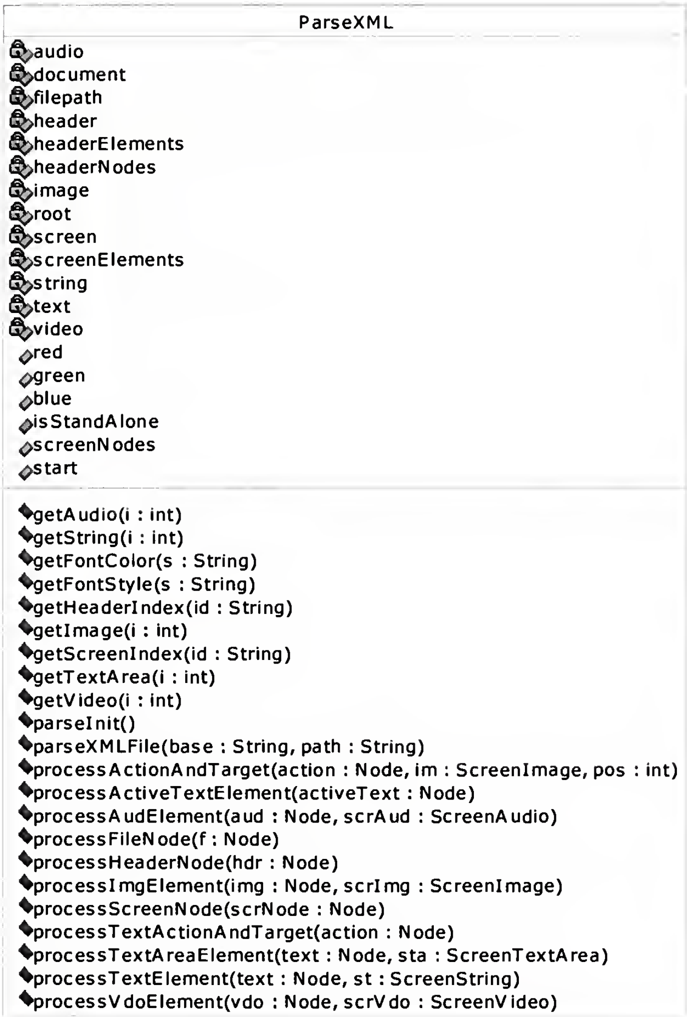
- *refreshString()*: Corresponds to the <text> tags in the XML file. Used for display of text on the screen using instances of StringJEditorPane class. It sets the font, size, color and location for each instance of StringJEditorPane and initializes it with the text to be displayed. Besides it also populates the instance with any actions to be performed (corresponding to <action> tags).
- *refreshImage()*: Corresponds to the <img> tag in the XML file. Used for display of images on the screen using instances of ImageJLabel class. It sets the size for each instance, creates an instance of *ImageIcon* initiated with the path for the image file and adds that instance of *ImageIcon* to the instance of ImageJLabel. Besides it also populates the instance with any actions to be performed (corresponding to <action> tags). Most popular image file types supported are .bmp, .gif and .jpg.
- *refreshTextArea()*: Corresponds to the <textArea> tags in the XML file. Used for display of documents on the screen using instances of *JEditorPane* class. Each instance is initialized with the path of the document and assigned its size and location on the screen. Also the instance is associated with an instance of *JScrollPane* class which facilitates the scrolling of document on the screen. The type of documents that can be displayed are .doc, .rtf, .htm and .txt. The version of HTML supported is 3.0 so HTML tags corresponding to that version will only

be correctly interpreted. Moreover, the HTML files cannot be linked, so in one instance only one HTML file can be displayed.

After these methods initialize the appropriate objects, the *jbInit()* method adds those contents to the applet and it is ready to be displayed. The browsing between screens is handled by each individual objects through their implementation of event listener method(s) in conjunction with the KioskEngine class. The applet keeps on running until it either the browser/frame is closed, in the event of which, the applet's *stop()* and *destroy()* methods are called in that order.

7.3.2 ParseXML class

Figure 9 presents the class diagram of the ParseXML Class.



ParseXML

audio
document
filepath
header
headerElements
headerNodes
image
root
screen
screenElements
string
text
video
red
green
blue
isStandAlone
screenNodes
start

getAudio(i : int)
getString(i : int)
getFontColor(s : String)
getFontStyle(s : String)
getHeaderIndex(id : String)
getImage(i : int)
getScreenIndex(id : String)
getTextArea(i : int)
getVideo(i : int)
parseInit()
parseXMLFile(base : String, path : String)
processActionAndTarget(action : Node, im : ScreenImage, pos : int)
processActiveTextElement(activeText : Node)
processAudElement(aud : Node, scrAud : ScreenAudio)
processFileNode(f : Node)
processHeaderNode(hdr : Node)
processImgElement(img : Node, scrImg : ScreenImage)
processScreenNode(scrNode : Node)
processTextActionAndTarget(action : Node)
processTextAreaElement(text : Node, sta : ScreenTextArea)
processTextElement(text : Node, st : ScreenString)
processVdoElement(vdo : Node, scrVdo : ScreenVideo)

Figure 9. Class Diagram of the ParseXML Class

This class plays a key role in the overall execution of the application and serves for one of the main purpose in the application, which is parsing the XML file, without which the application would not initiate and hence not work. The important packages (which provide with built in parser classes) imported by this class are *org.w3c.dom*, *org.xml.sax* and *javax.xml.parsers*.

Following is a brief about important methods in the class:

- *parseXML()*: The class parses the XML file based on DOM (Document Object Model) and obtains a document from the XML file. This document object is not much useful as it is, but it helps to identify and extract useful tag information from the XML document and provides useful methods for that purpose.
- *parseInit()*: After obtaining the document this method is used to parse the first screen/header pair from the document.
- *processScreenNode()* and *processHeaderNode()*: The method for processing a screen calls the method for processing its associated header (specified by header attribute in XML). Functionally in both methods is same as both screen and header can have the same types of contents inside it. Both methods use five different methods for processing each of the five multimedia objects that can appear in the XML file. The discussion about those five methods follows below.
- *processTextElement()*: This method corresponds to <text> tags in the XML file. It extracts attribute values, node contents and information from action/target tags (if present) from the current text node passed as an argument and stores the data in an instance of ScreenString class.
- *processTextAreaElement()*: This method corresponds to <textArea> tags in the XML file. It extracts attribute values from the current textArea node passed as an argument and stores the data in an instance of ScreenTextArea class.

- *processImgElement()*: This method corresponds to <img> tags in the XML file. It extracts attribute values and information from action/target tags (if present) for the current image node and stores the data in an instance of ScreenImage class.
- *processAudElement()*: This method corresponds to <aud> tags in the XML file. It extracts attribute values and information from file tags for the current audio node and stores the data in an instance of ScreenAudio class.
- *processVdoElement()*: This method corresponds to <vdo> tags in the XML file. It extracts attribute values and information from file tags for the current audio node and stores the data in an instance of ScreenVideo class.
- *getScreenIndex()* and *getHeaderIndex()*: At runtime, during the occurrence of an event these methods are used to identify the index value of the screen/header to be processed based on its id value. That id is then passed as an argument to processScreenNode() and processHeaderNode().

At any given time there parseXML can process only one screen/header pair and hence the multimedia objects contained only in that screen/header pair will be stored and will be accessible.

7.3.3 StringJEditorPane class

Figure 10 presents the class diagram of the StringJEditorPane Class:

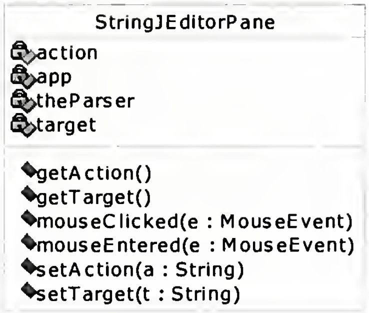

StringJEditorPane
action
app
theParser
target
getAction()
getTarget()
mouseClicked(e : MouseEvent)
mouseEntered(e : MouseEvent)
setAction(a : String)
setTarget(t : String)

Figure 10. Class Diagram of the StringJEditorPane Class

This class extends directly from *javax.swing.JEditorPane* and it implements two interfaces, namely *MouseListener* and *MouseMotionListener*, from the *java.awt.swing* package which helps to track and respond to mouse motion and actions. Instances of this class is used by the KioskEngine class and in turn each instance of this class will know the about the corresponding instance of KioskEngine class which is using it.

JEditorPane is a type of text component which can display formatted text according to specifications provided. By extending this class, an instance of StringJEditorPane will also be a text component and will inherit all the useful methods from the *JEditorPane* class that provide facility to set the font, style, size, color, etc. for the text component.

By implementing the two interfaces this class becomes a listener for itself which means that it would not be required to add instances of listeners to an instance of this class. One of the important methods from the *MouseListener* interface that this class implements is *mouseClicked()*, which is called by the JRE at runtime when a mouse is clicked on this text component. Notice that only those text components would respond to mouse actions for which there is a corresponding <action>/<target> pair of tags within the <text> tags; since it is not required that all text components should have an associated action/target.

Following is a pseudo code for the *mouseClicked()* method:

1. Get the instance of ParseXML currently used by the KioskEngine class.
2. Use the ParseXML instance to find the index of the screen id specified within <target> tags.
3. Use the ParseXML instance to parse the particular screen (with its associated header).
4. Stop any previous audio/video players from previous screen still being played.
5. Use the instance of KioskEngine class to display the multimedia objects on the target screen.

7.3.4 ImageJLabel class

Figure 11 presents the class diagram of the ImageJLabel Class.

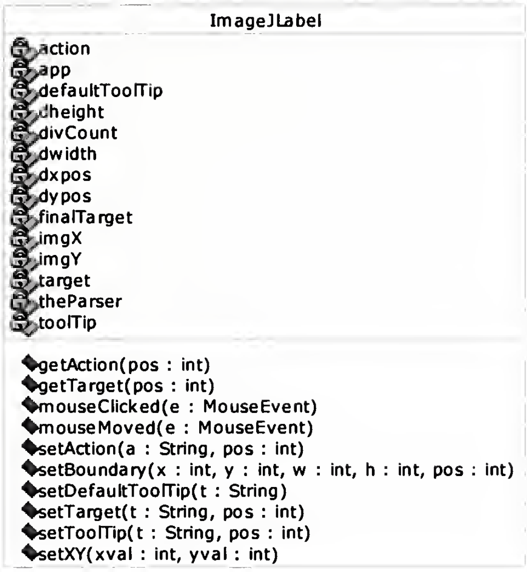

ImageJLabel
action
app
defaultToolTip
dheight
divCount
dwidth
dxpos
dypos
finalTarget
imgX
imgY
target
theParser
toolTip
getAction(pos : int)
getTarget(pos : int)
mouseClicked(e : MouseEvent)
mouseMoved(e : MouseEvent)
setAction(a : String, pos : int)
setBoundary(x : int, y : int, w : int, h : int, pos : int)
setDefaultToolTip(t : String)
setTarget(t : String, pos : int)
setToolTip(t : String, pos : int)
setXY(xval : int, yval : int)

Figure 11. Class Diagram of the ImageJLabel Class

This class extends directly from *javax.swing.JLabel* class and it implements two interfaces, namely *MouseListener* and *MouseMotionListener*, from the *java.awt.swing* package which helps to track and respond to mouse motion and actions. Instances of this class is used by the KioskEngine class and in turn each instance of this class will know the about the corresponding instance of KioskEngine class which is using it.

JLabel is a component which can display text as well as images. Although displaying images is not directly supported by *JLabel*, however, images can be embedded in objects of another class called *ImageIcon.* For displaying an image, the KisokEngine creates an instance of *ImageIcon* and initializes it with the source path of the image file and then adds this instance to the instance of ImageJLabel.

By implementing the two interfaces this class becomes a listener for itself which means that it would not be required to add instances of listeners to an instance of this class. It implements two important methods, *mouseClicked()* and *mouseMoved()* from interfaces *MouseListener* and *MouseMotionListener*, respectively. The implementation of *mouseMoved()* method allows to support the functioning of image maps to be created and work properly. It is because of this method that a single <img> tag (in the XML file) can contain lot of <action> tags in order to respond differently based on which part of the image the action occurred. This method is called every time the mouse cursor moves, and based on the movement of the mouse cursor this method changes the tool-tip for the image and internally the target to respond with.

Following is a pseudo code for the *mouseMoved()* method:

1. Get the X and Y co-ordinates for the current position of the mouse cursor.
2. For each rectangular area defined in the XML for this image, check to see if the cursor is inside any of the rectangle. If it is inside a rectangle change the cursor to hand-shaped, set the tool-tip for that rectangle and set the corresponding target. If its not inside any rectangle set default cursor and tool-tip.

The method *mouseClicked()* is called by the JRE at runtime when a mouse is clicked on this image component. Notice that only those image components would respond to mouse actions for which there is/are corresponding <action>/<target> pair of tags within the <img> tags; since it is not required that all image components should have an associated action/target.

Following is a pseudo code for the *mouseClicked()* method:

1. Get the instance of ParseXML currently used by the KioskEngine class.
2. Use the ParseXML instance to find the index of the screen id specified within <target> tags.
3. Use the ParseXML instance to parse the particular screen (with its associated header).
4. Stop any previous audio/video players from previous screen still being played.
5. Use the instance of KioskEngine class to display the multimedia objects on the target screen.

7.3.5 AudioCombo class

Figure 12 presents the class diagram of the AudioCombo Class.

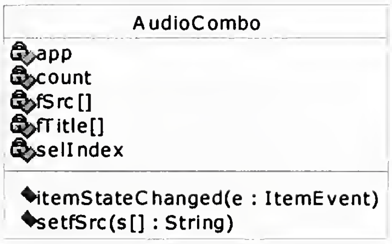


Figure 12. Class Diagram of the AudioCombo Class

This class extends directly from *javax.swing.JcomboBox*. By implementing the *ItemListener* interface this class becomes a listener for itself which means that it would not be required to add instances of listeners to an instance of this class. An instance of this class is created by the KioskEngine class for each occurance of the audio player. Each instance is populated with the list of sound files for that corresponding player. The sound files are displayed in the combo-box in the order in which they are present in the XML file.

To change the sound file being played the user can select the file from the drop-down combo-box. To support switching between different sound files this class implements the method *itemStateChanged()* of the *ItemListener* interface.

Following is the pseudo code for the *itemStateChanged()* method:

1. Get the index of the sound file selected.
2. Refresh the audio player instance with the file source of selected index.
3. Start the audio player.

7.3.6 VideoCombo class

Figure 13 presents the class diagram of the VideoCombo Class.

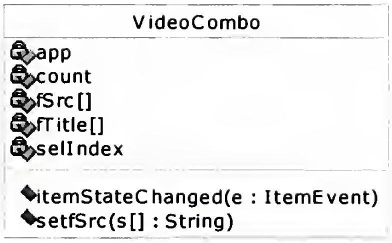


Figure 13. Class Diagram of the VideoCombo Class

This class extends directly from *javax.swing.JcomboBox.* By implementing the *ItemListener* interface this class becomes a listener for itself which means that it would not be required to add instances of listeners to an instance of this class. An instance of this class is created by the KioskEngine class for each occurance of the video player. Each instance is populated with the list of video files for that corresponding player. The video files are displayed in the combo-box in the order in which they are present in the XML file.

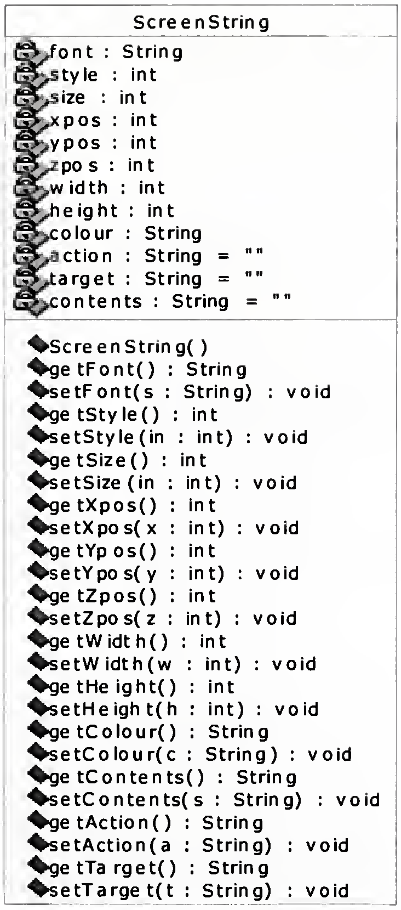
To change the video file being played the user can select the file from the drop-down combo-box. To support switching between different video files this class implements the method *itemStateChanged()* of the *ItemListener* interface.

Following is the pseudo code for the *itemStateChanged()* method:

```
1. Get the index of the video file selected.
2. Refresh the video player instance with the file source of selected index.
3. Clear area and start the video player.
```

7.3.7 ScreenString class

Figure 14 presents the class diagram of the ScreenString Class.



ScreenString
font : String
style : int
size : int
xpos : int
ypos : int
zpos : int
width : int
height : int
colour : String
action : String = ""
target : String = ""
contents : String = ""

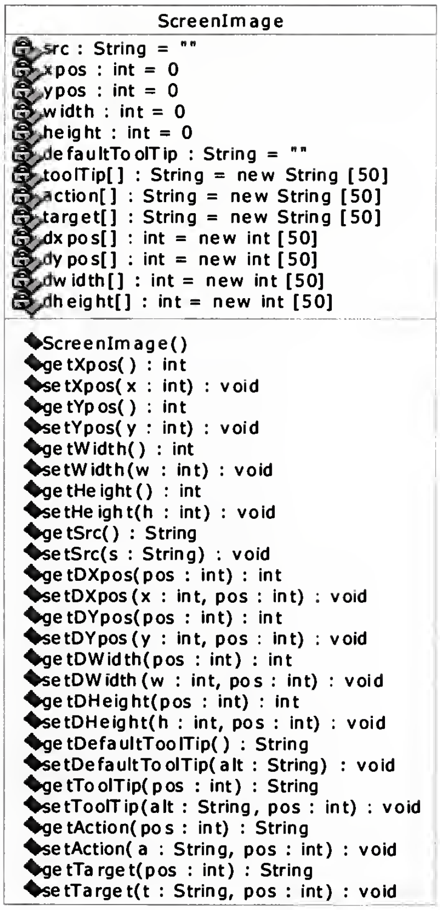
ScreenString()
getFont() : String
setFont(s : String) : void
getStyle() : int
setStyle(in : int) : void
getSize() : int
setSize(in : int) : void
getXpos() : int
setXpos(x : int) : void
getYpos() : int
setYpos(y : int) : void
getZpos() : int
setZpos(z : int) : void
getWidth() : int
setWidth(w : int) : void
getHeight() : int
setHeight(h : int) : void
getColour() : String
setColour(c : String) : void
getContents() : String
setContents(s : String) : void
getAction() : String
setAction(a : String) : void
getTarget() : String
setTarget(t : String) : void

Figure 14. Class Diagram of the ScreenString Class

This class is essentially a data structure for holding string objects corresponding to <text> tags inside the XML file. It has private data members which store the attributes specified in the <text> tag such as font, style, size, color, position and associated action/target for this text. The class contains get() and set() methods for accessing each of the attributes. An instance of this class is created by the ParseXML class for each occurrence of <text> tag in the XML file and initialized with the attributes. Those instances are then used by the KioskEngine class and the attribute values are accessed from the data members for displaying the text on the screen.

7.3.8 ScreenImage class

Figure 15 presents the class diagram of the ScreenImage Class.



```
ScreenImage
src : String = ""
xpos : int = 0
ypos : int = 0
width : int = 0
height : int = 0
defaultToolTip : String = ""
toolTip[] : String = new String [50]
action[] : String = new String [50]
target[] : String = new String [50]
dxpos[] : int = new int [50]
dypos[] : int = new int [50]
dwidth[] : int = new int [50]
dheight[] : int = new int [50]

ScreenImage()
getXpos() : int
setXpos(x : int) : void
getYpos() : int
setYpos(y : int) : void
getWidth() : int
setWidth(w : int) : void
getHeight() : int
setHeight(h : int) : void
getSrc() : String
setSrc(s : String) : void
getDXpos(pos : int) : int
setDXpos(x : int, pos : int) : void
getDYpos(pos : int) : int
setDYpos(y : int, pos : int) : void
getDWidth(pos : int) : int
setDWidth(w : int, pos : int) : void
getDHeight(pos : int) : int
setDHeight(h : int, pos : int) : void
getDefaultToolTip() : String
setDefaultToolTip(alt : String) : void
getToolTip(pos : int) : String
setToolTip(alt : String, pos : int) : void
getAction(pos : int) : String
setAction(a : String, pos : int) : void
getTarget(pos : int) : String
setTarget(t : String, pos : int) : void
```

Figure 15. Class Diagram of the ScreenImage Class

This class is essentially a data structure for holding image objects corresponding to <img> tags inside the XML file. It has private data members which store the attributes specified in the <img> tag such as source, alt, size, position and associated action/target(s) for this image. The class contains get() and set() methods for accessing each of the attributes. An instance of this class is created by the ParseXML class for each occurrence of <img> tag in the XML file and initialized with the attributes. Those instances are then used by the KioskEngine class and the attribute values are accessed from the data members for displaying the image on the screen.

7.3.9 ScreenTextArea class

Figure 16 presents the class diagram of the ScreenTextArea Class:

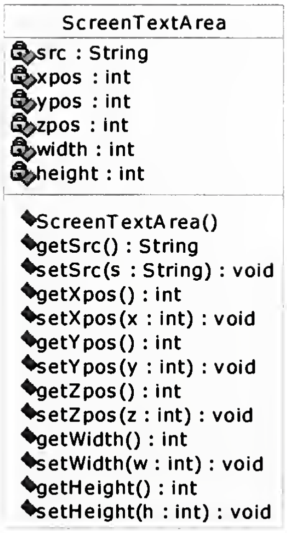
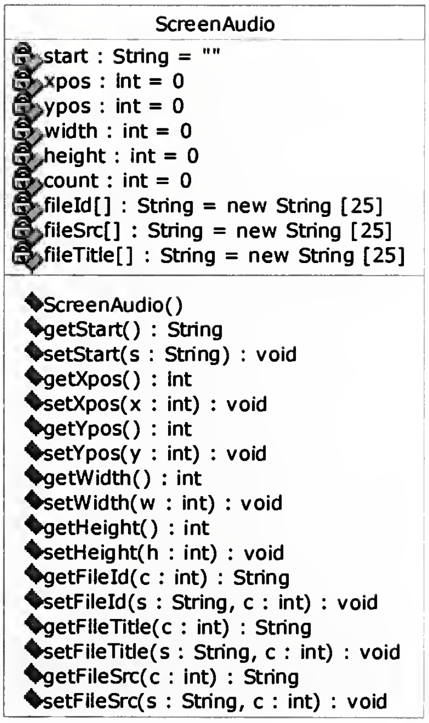

ScreenTextArea
src : String
xpos : int
ypos : int
zpos : int
width : int
height : int
ScreenTextArea()
getSrc() : String
setSrc(s : String) : void
getXpos() : int
setXpos(x : int) : void
getYpos() : int
setYpos(y : int) : void
getZpos() : int
setZpos(z : int) : void
getWidth() : int
setWidth(w : int) : void
getHeight() : int
setHeight(h : int) : void

Figure 16. Class Diagram of the ScreenTextArea Class

This class is essentially a data structure for holding document objects corresponding to <textArea> tags inside the XML file. It has private data members which store the attributes specified in the <textArea> tag such as source, size and position. The class contains get() and set() methods for accessing each of the attributes. An instance of this class is created by the ParseXML class for each occurrence of <textArea> tag in the XML file and initialized with the attributes. Those instances are then used by the KioskEngine class and the attribute values are accessed from the data members for displaying the document on the screen.

7.3.10 ScreenAudio class

Figure 17 presents the class diagram of the ScreenAudio Class.



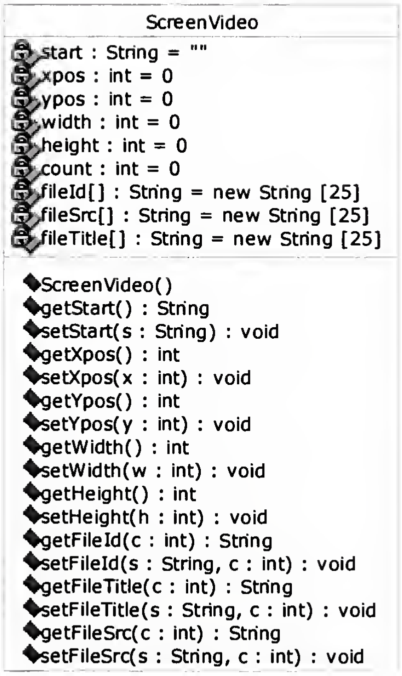
```
ScreenAudio
start : String = ""
xpos : int = 0
ypos : int = 0
width : int = 0
height : int = 0
count : int = 0
fileId[] : String = new String [25]
fileSrc[] : String = new String [25]
fileTitle[] : String = new String [25]

ScreenAudio()
getStart() : String
setStart(s : String) : void
getXpos() : int
setXpos(x : int) : void
getYpos() : int
setYpos(y : int) : void
getWidth() : int
setWidth(w : int) : void
getHeight() : int
setHeight(h : int) : void
getFileId(c : int) : String
setFileId(s : String, c : int) : void
getFileTitle(c : int) : String
setFileTitle(s : String, c : int) : void
getFileSrc(c : int) : String
setFileSrc(s : String, c : int) : void
```

Figure 17. Class Diagram of the ScreenAudio Class

This class is essentially a data structure for holding audio objects corresponding to <aud> tags inside the XML file. It has private data members which store the attributes specified in the <aud> tag such as start file, size, position and associated sound files for this text described by <file> tags. The class contains get() and set() methods for accessing each of the attributes. An instance of this class is created by the ParseXML class for each occurrence of <aud> tag in the XML file and initialized with the attributes. Those instances are then used by the KioskEngine class and the attribute values are accessed from the data members for displaying the audio player on the screen.

7.3.11 ScreenVideo class

Figure 18 presents the class diagram of the ScreenVideo Class.



```
ScreenVideo
start : String = ""
xpos : int = 0
ypos : int = 0
width : int = 0
height : int = 0
count : int = 0
fileId[] : String = new String [25]
fileSrc[] : String = new String [25]
fileTitle[] : String = new String [25]

ScreenVideo()
getStart() : String
setStart(s : String) : void
getXpos() : int
setXpos(x : int) : void
getYpos() : int
setYpos(y : int) : void
getWidth() : int
setWidth(w : int) : void
getHeight() : int
setHeight(h : int) : void
getFileId(c : int) : String
setFileId(s : String, c : int) : void
getFileTitle(c : int) : String
setFileTitle(s : String, c : int) : void
getFileSrc(c : int) : String
setFileSrc(s : String, c : int) : void
```

Figure 18. Class Diagram of the ScreenVideo Class

This class is essentially a data structure for holding video objects corresponding to <vdo> tags inside the XML file. It has private data members which store the attributes specified in the <vdo> tag such as start file, size, position and associated video files for this text described by <file> tags. The class contains get() and set() methods for accessing each of the attributes. An instance of this class is created by the ParseXML class for each occurrence of <vdo> tag in the XML file and initialized with the attributes. Those instances are then used by the KioskEngine class and the attribute values are accessed from the data members for displaying the video player on the screen.

8. Proof of Concept

This section is focused on providing useful hints for putting together a kiosk using KioskEngine followed by is an illustration of the idea applied to a pilot application.

8.1 Useful Hints for putting together a Kiosk using KioskEngine

When the target kiosk is intended to run in a browser then it is required to create an HTML file which embeds an applet within itself and calls the KioskEngine class which should be placed (along with other classes) in kioskengine folder. Also the path of the xml file to be used by KioskEngine class has to be passed as a parameter to the applet.

When the target kiosk is intended to run standalone then the classes can be packaged in a .jar file. In a standalone mode the packaged JAR application will use the kiosk.xml file from the xml folder. This restriction is due to the filename already specified in the KioskEngine class. The (double-click able) JAR application should be placed in the parent folder of the folder in which the xml file is placed.

For best results it is useful to create a folder for each of the media types and for xml file. The HTML or JAR application should be placed in the same folder as the folders for the media and xml files. For example, assume that a kiosk demo application called DEMO is created. Then the DEMO folder structure would look as below:

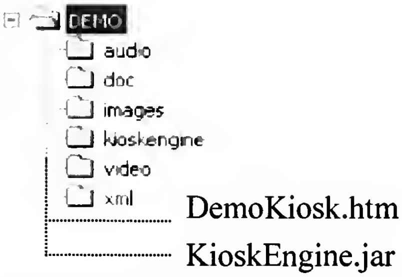
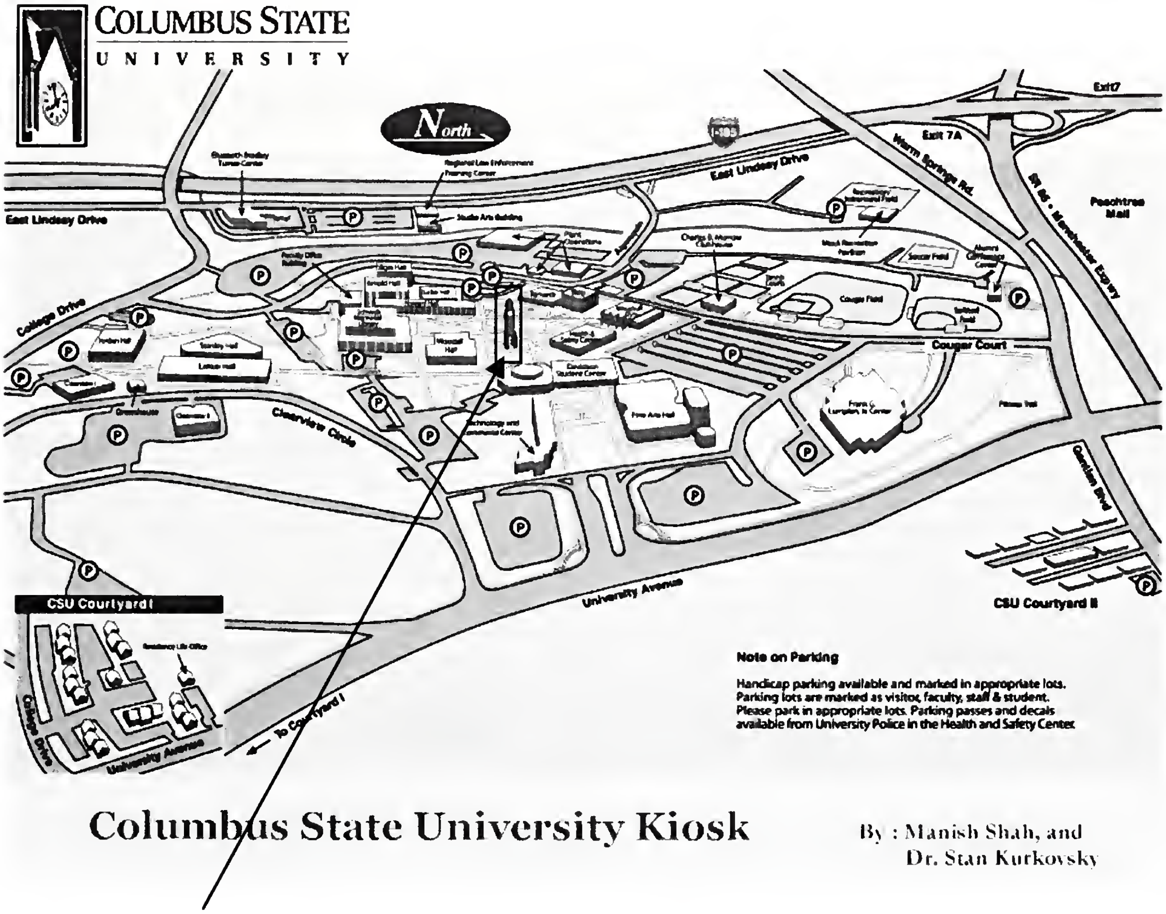


Figure 19. Folder Structure of a DEMO Kiosk Application

8.2 Pilot Application : Columbus State University (CSU) Kiosk

This sections discusses the pilot application titled "Columbus State University Kiosk" which was developed in order to demonstrate a working model and the functionality of KioskEngine. The kiosk intends to provide information about the map and individual buildings on campus of the Columbus State University to the users.

The start screen consists of the map of Columbus State University and looks as below:


COLUMBUS STATE
UNIVERSITY
North
East Lindsey Drive
East Lindsey Drive
College Drive
Clearview Circle
University Avenue
Cougar Court
Exit 7
Exit 7A
Warm Springs Rd.
SR 85 - Manchester Expwy
Peachtree Mall
Gentian Blvd
CSU Courtyard I
CSU Courtyard II
College Drive
University Avenue
To Courtyard I

Note on Parking

Handicap parking available and marked in appropriate lots. Parking lots are marked as visitor, faculty, staff & student. Please park in appropriate lots. Parking passes and decals available from University Police in the Health and Safety Center.

Columbus State University Kiosk

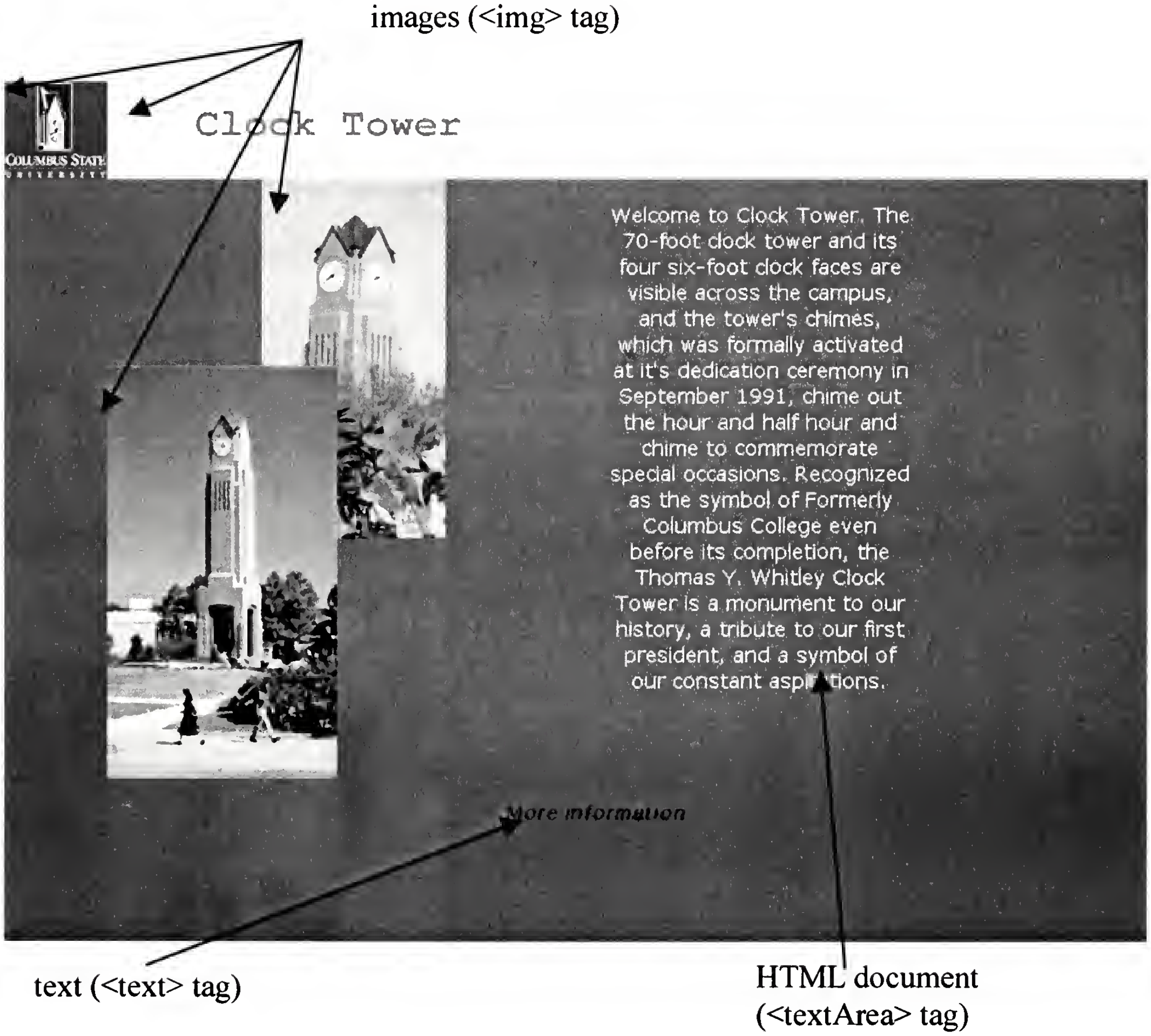
By : Manish Shah, and
Dr. Stan Kurkovsky

hot spot : Clock Tower

Figure 20. Snapshot of Start Screen of the CSU Kiosk

The above start screen has some hot spots (clickable regions) corresponding to different buildings on the map. When a user clicks on a hot spot the kiosk displays the screen corresponding to that particular building.

When the user clicks on the hot spot corresponding to the "Clock Tower" as shown in the previous figure the screen corresponding to "Clock Tower" is displayed and looks as below:



Clock Tower

Welcome to Clock Tower. The 70-foot clock tower and its four six-foot clock faces are visible across the campus, and the tower's chimes, which was formally activated at it's dedication ceremony in September 1991, chime out the hour and half hour and chime to commemorate special occasions. Recognized as the symbol of Formerly Columbus College even before its completion, the Thomas Y. Whitley Clock Tower is a monument to our history, a tribute to our first president, and a symbol of our constant aspirations.

More information

Figure 21. Snapshot of Tower Screen of the CSU Kiosk

The image on the top left corner takes user back to the main map and the text on the bottom takes to further information about "Clock Tower". Similarly there are number of different screens, each corresponding to a particular building on the map. The XML file for this kiosk is included as an appendix towards the end.

9. Extensibility of KioskEngine

One of the immediate enhancements that can be made to the KioskEngine is to develop a GUI-based editor which would allow users to select types of media for a screen and drag and drop them on the screen on the desired location. The location for that object should be detected by the editor and the attributes and information that it is supposed to display should be easily editable using the property-list of that object. The editor would in effect just build the XML file for the kiosk based on the design made by the user and not store any graphical information for the kiosk. To allow making changes to a kiosk, the editor can use KioskEngine to display how the current kiosk looks like and then save the changes to the XML file based on the changes made by the user in the GUI editor. Such a tool would help users to build kiosks quickly and easily.

Another add-on feature that is lucrative is to allow wireless support for kiosks. For this to work, the KioskEngine can be modified such that it extracts information from the XML file and creates corresponding WML (Wireless Markup Language) file(s) for the kiosk that would essentially display textual information instead of graphical information for each screen of the kiosk. The WML files for the kiosk can be hosted on a WAP gateway and can be configured to allow particular group of users to access the kiosk through their wireless devices such as PDA, Pocket PC, etc.

10. Conclusion

Certainly, KioskEngine is a very powerful multifaceted light-weight platform-independent application, which can be used to run a variety of multimedia rich standalone or web-enabled kiosks on various software and hardware platforms. It acts as a driver program behind a kiosk and the program does not need to be changed from one kiosk to another. The structure of the kiosk is provided in the form of XML file as an input to the KioskEngine and the data for the kiosk is completely separate from the program. As a result of this, the structure of the kiosk can be modified or exported at any time and so can be the multimedia data, both independent of each other. Moreover, various additional functionalities and features can be added to the program to make it even more powerful.

References

[Apunix03] The Apunix Kiosk Engine for the Java ® Platform, ©Apunix Computer Services, 2003, Website: http://www.apunix.com/kiosk_products/software.html

[Douglas00] Retail kiosks: Breaking new ground, By Mitchell Douglas, 12 Dec 2000, Website: http://www.kioskmarketplace.com/research.htm

[LindholmY03] The Java™ Virtual Machine Specification, 2nd Edition, By Tim Lindholm and Frank Yellin, ©Sun Microsystems, Inc., 1999, Website: http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html

[MacroFlash03] Macromedia Flash MX, ©Macromedia, Inc., 2003, Website: http://www.macromedia.com/software/flash/

[MacroVid03] Using video in Macromedia Flash MX, ©Macromedia, Inc., 2003 Website:http://www.macromedia.com/support/flash/images_video/flash_video/

[MassMlt03] Kiosk-In-A-Box Software, ©Mass Multimedia, Inc., 2003, Website: http://www.touchscreens.com/kbox-full.html

[MS03] Microsoft® Corporation, Microsoft®, 2003, Website: http://www.microsoft.com/

[MSDN03] Microsoft® Developer Network Home, Microsoft® Corporation, 2003, Website: http://msdn.microsoft.com

[SunJ2SE03] Java™ 2 Platform, Standard Edition (J2SE™), ©Sun Microsystems, 2003, Website: http://java.sun.com/j2se/

[SunJMF03] Java™ Media Framework API, ©Sun Microsystems, Inc., 2003, Website: http://java.sun.com/products/java-media/jmf/index.html

[SunSwing03] Creating a GUI with JFC/Swing, ©Sun Microsystems, Inc., 2003 Website: http://java.sun.com/docs/books/tutorial/uiswing/

[Touch03] KioskEngine™ software, ©Touch Controls, Inc., 2003, Website: http://www.publicaccesskiosks.com/pak/kioskengine.htm

Appendix: XML file for Columbus State University Kiosk

```
<?xml version = "1.0" ?>
<kiosk red="0" green="102" blue="204" start="csumain">


        <header id="csuheader">
                <img src="images/csu/csu_logo.gif" alt="CSU MAP" xpos="0" ypos="0" width="72"
                height="68">
                        <action type="onclick" alt="CSU MAP" dxpos="0" dypos="0" dwidth="72"
                        dheight="68">
                                <target>csumain</target>
                        </action>
                </img>

        </header>

        <screen id="csumain" header="">
                <img src="images/csu/kioskNameplate.jpg" alt="Columbus State University Kiosk"
                xpos="0" ypos="531" width="800" height="69">
                </img>

                <img src="images/csu/map80.jpg" alt="CSU MAP" xpos="0" ypos="0" width="800"
                height="531">
                        <action type="onclick" alt="About Columbus State University" dxpos="60"
                        dypos="0" dwidth="181" dheight="45">
                                <target>aboutcsu</target>
                        </action>
                        <action type="onclick" alt="Clock Tower" dxpos="344" dypos="201"
                        dwidth="13" dheight="37">
                                <target>clocktower1</target>
                        </action>
                        <action type="onclick" alt="Simon Schwob Library" dxpos="232" dypos="215"
                        dwidth="51" dheight="25">
                                <target>library1</target>
                        </action>
                        <action type="onclick" alt="Lenoir Hall" dxpos="107" dypos="243"
                        dwidth="79" dheight="18">
                                <target>lenoir1</target>
                        </action>
                        <action type="onclick" alt="Richards Hall" dxpos="356" dypos="195"
                        dwidth="55" dheight="15">
                                <target>richards1</target>
                        </action>
                        <action type="onclick" alt="Howard Hall" dxpos="256" dypos="202"
                        dwidth="70" dheight="12">
                                <target>howard1</target>
                        </action>
                        <action type="onclick" alt="Woodall Hall" dxpos="293" dypos="229"
                        dwidth="33" dheight="18">
                                <target>woodall1</target>
                        </action>
                        <action type="onclick" alt="Davidson Student Center" dxpos="343"
                        dypos="249" dwidth="80" dheight="11">
                                <target>davidson1</target>
                        </action>
                </img>

        </screen>

        <screen id="aboutcsu" header="csuheader">
```

```
		<img src="images/csu/aboutPlate.jpg" alt="Columbus State University Kiosk"
		xpos="72" ypos="0" width="728" height="68">
		</img>
		<textArea src="doc/csu/aboutCSU.htm" xpos="72" ypos="68" width="650"
		height="517">
		</textArea>

	</screen>
	<screen id="clocktower1" header="csuheader">
		<img src="images/csu/towerPlate.jpg" alt="Clock Tower" xpos="72" ypos="0"
		width="728" height="68">
		</img>
		<img src="images/csu/ClockTower1.jpg" alt="Clock Tower" xpos="72" ypos="198"
		width="161" height="288">
		</img>
		<img src="images/csu/towerinspring3.jpg" alt="Clock Tower in Spring" xpos="180"
		ypos="68" width="129" height="250">
		</img>
		<textArea src="doc/csu/tower1.htm" xpos="351" ypos="68" width="400"
		height="417">
		</textArea>
		<text font="Serif" style="BOLDITALIC" size="14" xpos="351" ypos="500" width="250"
		height="30" color="BLACK">More information...
			<action type="onclick">
				<target>clocktower2</target>
			</action>
		</text>
	</screen>

	<screen id="clocktower2" header="csuheader">
		<img src="images/csu/towerPlate.jpg" alt="Clock Tower" xpos="72" ypos="0"
		width="728" height="68">
		</img>
		<img src="images/csu/ClockTower1.jpg" alt="Clock Tower" xpos="72" ypos="198"
		width="161" height="288">
		</img>
		<img src="images/csu/towerinspring3.jpg" alt="Clock Tower in Spring" xpos="180"
		ypos="68" width="129" height="250">
		</img>
		<textArea src="doc/csu/tower2.htm" xpos="351" ypos="68" width="400"
		height="417">
		</textArea>
		<text font="Serif" style="BOLDITALIC" size="14" xpos="351" ypos="500" width="250"
		height="30" color="BLACK">Previous information...
			<action type="onclick">
				<target>clocktower1</target>
			</action>
		</text>
	</screen>
	<screen id="library1" header="csuheader">
		<img src="images/csu/libraryPlate.jpg" alt="Simon Schwob Library" xpos="72"
		ypos="0" width="728" height="68">
		</img>
		<img src="images/csu/schwob1.jpg" alt="Simon Schwob Library" xpos="72"
		ypos="212" width="279" height="171">
		</img>
		<img src="images/csu/library1.jpg" alt="Simon Schwob Library" xpos="246" ypos="68"
		width="279" height="171">
		</img>
		<textArea src="doc/csu/library1.htm" xpos="408" ypos="290" width="350"
		height="210">
		</textArea>
```

```xml
        <text font="Serif" style="BOLDITALIC" size="14" xpos="508" ypos="510" width="250"
        height="30" color="BLACK">More information...
                <action type="onclick">
                        <target>library2</target>
                </action>
        </text>
</screen>
<screen id="library2" header="csuheader">
        <img src="images/csu/libraryPlate.jpg" alt="Simon Schwob Library" xpos="72"
        ypos="0" width="728" height="68">
        </img>
        <img src="images/csu/schwob2.jpg" alt="Simon Schwob Library" xpos="373"
        ypos="68" width="231" height="120">
        </img>
        <img src="images/csu/girl_library.jpg" alt="Girl at Library" xpos="517" ypos="167"
        width="231" height="120">
        </img>
        <textArea src="doc/csu/library2.htm" xpos="72" ypos="221" width="400"
        height="255">
        </textArea>
        <text font="Serif" style="BOLDITALIC" size="14" xpos="82" ypos="490" width="250"
        height="30" color="BLACK">Previous information...
                <action type="onclick">
                        <target>library1</target>
                </action>
        </text>
</screen>
<screen id="richards1" header="csuheader">
        <img src="images/csu/richardsPlate.jpg" alt="Richards Hall" xpos="72" ypos="0"
        width="728" height="68">
        </img>
        <img src="images/csu/richardspicture.jpg" alt="Richards Picture" xpos="162"
        ypos="68" width="120" height="156">
        </img>
        <img src="images/csu/richards3.jpg" alt="Richards entrance" xpos="50" ypos="210"
        width="120" height="156">
        </img>
        <img src="images/csu/richards1.jpg" alt="Richards Hall" xpos="440" ypos="68"
        width="245" height="156">
        </img>
        <img src="images/csu/richards2.jpg" alt="Richards Hall" xpos="524" ypos="210"
        width="245" height="156">
        </img>
        <textArea src="doc/csu/richards1.htm" xpos="222" ypos="229" width="300"
        height="360">
        </textArea>
</screen>

<screen id="lenoir1" header="csuheader">
        <img src="images/csu/lenoirPlate.jpg" alt="Lenoir Hall" xpos="72" ypos="0"
        width="728" height="68">
        </img>
        <img src="images/csu/lenoirplaque.jpg" alt="Lenoir Plaque" xpos="75" ypos="68"
        width="300" height="163">
        </img>
        <img src="images/csu/lenoirpicture.jpg" alt="Lenoir Picture" xpos="0" ypos="196"
        width="138" height="208">
        </img>
        <img src="images/csu/lenoir1.jpg" alt="Lenoir Hall" xpos="500" ypos="241"
        width="300" height="163">
        </img>
```

```
                <img src="images/csu/lenoir2.jpg" alt="Lenoir Hall" xpos="376" ypos="68"
                width="300" height="163">
                </img>
                <textArea src="doc/csu/lenoir1.htm" xpos="152" ypos="252" width="335"
                height="325">
                </textArea>
        </screen>
        <screen id="howard1" header="csuheader">
                <img src="images/csu/howardPlate.jpg" alt="Howard Hall" xpos="72" ypos="0"
                width="728" height="68">
                </img>
                <img src="images/csu/howard1.jpg" alt="Howard Hall" xpos="268" ypos="68"
                width="246" height="153">
                </img>
                <img src="images/csu/howard2.jpg" alt="Howard Hall" xpos="72" ypos="288"
                width="246" height="153">
                </img>
                <img src="images/csu/howard3.jpg" alt="Howard Hall" xpos="465" ypos="288"
                width="246" height="153">
                </img>

        </screen>
        <screen id="woodall1" header="csuheader">
                <img src="images/csu/woodallPlate.jpg" alt="Woodall Hall" xpos="72" ypos="0"
                width="728" height="68">
                </img>
                <img src="images/csu/woodall1.jpg" alt="Woodall Hall" xpos="72" ypos="68"
                width="243" height="389">
                </img>
                <textArea src="doc/csu/woodall1.htm" xpos="316" ypos="68" width="450"
                height="417">
                </textArea>
        </screen>
        <screen id="davidson1" header="csuheader">
                <img src="images/csu/davidsonPlate.jpg" alt="Davidson Student Center" xpos="72"
                ypos="0" width="728" height="68">
                </img>
                <img src="images/csu/davidson1.jpg" alt="Davidson Student Center" xpos="72"
                ypos="68" width="375" height="157">
                </img>
                <img src="images/csu/davidson2.jpg" alt="Davidson Student Center" xpos="550"
                ypos="68" width="250" height="400">
                </img>
                <textArea src="doc/csu/davidson1.htm" xpos="122" ypos="226" width="375"
                height="417">
                </textArea>
        </screen>

</kiosk>
```