WEB APPLICATION TOOL

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Aditi A. Ghode

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

December, 2007

Web Application Tool

By Aditi A. Ghode

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

_____ _____

Dr. Kasi Periyasamy — Date
Examination Committee Chairperson

_____ _____

Dr. David Riley — Date
Examination Committee Member

_____ _____

Dr. Mao Zheng — Date
Examination Committee Member

ABSTRACT

Ghode, Aditi, A., “Web Application Tool”, Master of Software Engineering, December 2007, Advisors: Dr. David Riley, Dr. Kasi Periyasamy.

Web site development is a difficult endeavor due to the complexities of programming languages and maintenance cost, especially for those who do not have sufficient computer background. This project was to develop a tool that makes web site management more convenient for non-sophisticated users. With the help of this tool anyone can create web pages and manage a website with no technical expertise or knowledge of HTML. The specific purpose of the work was to make the web more accessible to the people who work for non-profit organization, to allow them to publish websites without knowledge of HTML tools and to broaden the base of their work on the web. The project started with the aim to develop an “Easy to Use – WYSIWYG” tool. This manuscript describes the development process for the tool including the activities performed in each phase of the development process, the challenges encountered, the issues that arose, the current status of the project, and its features with continuing work. An important part in the design of the GUI for this tool is that it be easily used by those familiar with Microsoft Word.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who gave me the possibility to complete this project. I am deeply indebted to my project advisors Dr. David Riley and Dr. Kasi Periyasamy for their valuable guidance.

Furthermore, I thank the project sponsor, World Services of Lacrosse who initiated this project and provided the requirements for this project. This work also benefited from discussions with my project advisors and customers.

I would also like to express my thanks to the Computer Science Department and the University of Wisconsin-La Crosse for providing the computing environment for my project.

Finally, I wish to thank my husband Ashish for his patience and encouragement during the tenure of this project. Especially, I would like to give my special thanks to our baby girl Ashita whose support and patient love enabled me to complete this work.

TABLE OF CONTENTS

LIST OF TABLES

Tables **Page**

LIST OF FIGURES

GLOSSARY

CSS

In web development, **Cascading Style Sheets** (**CSS**) is a stylesheet language used to describe the presentation of a document written in a HTML. In WAT, CSS is used to help readers of web pages to define colors, fonts, layout, and other aspects of document presentation.

Cygwin

Cygwin is a Linux-like environment for Windows.

Directory

Directory is a way of organizing files and other directories. It is eequivalent to a ‘folder’ in Windows.

DOM

The Document Object Model is an API for HTML and XML documents.

DTD

DTD is an acronym of Document type Definition. It is one of several SGML and XML schema languages, and is also the term used to describe a document.

FTP

FTP or **File Transfer Protocol** is used to transfer data from one computer to another over the Internet, or through a network.

GUI

Graphical User Interface

HTML

HTML, an initialism of **Hypertext Markup Language**, is the predominant markup language for web pages. It provides a means to describe the structure of text-based information in a document — by denoting certain text as headings, paragraphs, lists, and so on — and to supplement that text with *interactive forms*, embedded *images*, and other objects.

IEEE

Institute of Electrical and Electronics Engineers. The IEEE (Institute of Electrical and Electronics Engineers, Inc.) is the world's leading professional association for the advancement of technology.

IDL

An Interface Description Language (IDL) (or alternately, interface definition language) is a specification language used to describe a software component's interface.

ISP

Internet Service Provider is a business or organization that provides consumers or businesses access to the Internet and related services.

MSAA

Microsoft Active Accessibility is a COM(Component Object Model)-based technology designed to improve the way accessibility aids work with applications running on Microsoft Windows. Accessibility aids may include screen readers for the visually impaired, visual indicators or captions for people with hearing loss, software to compensate for motion disabilities, etc.

Mozilla

This term used to refer to a number of similar browsers including Netscape 6 and later, Mozilla and Firefox.

Stylesheet

A text file, that may be a part of the page in use, that describes the position, appearance or behaviour of some or all of the HTML elements of which the page consists.

Tab (window)

Tab is a part of many window structures which emulate the tab on a paper filing system designed to give quick access to part of the file. When a tab is clicked a new display will appear within the same window.

Tab (key)

The keyboard key is intended to emulate the action of the tab key on a typewriter.

Upload (ing)

Upload is a process which transfer the web files from a local computer to the computer which will host them on the web.

URL

Uniform Resource Locator. The web address of an item.

W3C

World Wide Web Consortium http://www.w3.org. It is a consortium of the 500 biggest IT corporations who got together to define specifications and recommendations so that languages like HTML or CSS can inter-operate without problems on different platforms, devices, operating systems and media.

XBL

XML Binding Language

XPCOM

Cross-Platform Component Object Model

XPIDL

Cross Platform Interface Definition Language

XSLT

XSLT is the most important part of XSL. XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML.

XTF

eXtensible Text Framework

XUL

eXtensible User-interface Language. It is **Mozilla's** XML-based, cross-platform user-interface technology**.**

1. Introduction

Having a web site to promote a business is virtually imperative nowadays. Nearly all households have access to the internet, sometimes even on more than one computer. The great thing about the Internet is that it can reach the whole world and not just in the neighborhood. Websites are an effective method for exchanging cultures and ideas between people on the web. Websites can be shaped to meet the needs of almost any kind of organization.

The sponsor for this project - *World Services of La Crosse, Inc.* (WSL) - is a non-profit organization. WLS was organized in response to an interest in the community of La Crosse, Wisconsin, to continue the international development efforts of the former La Crosse International Health Partnership. The latter had contacts in Russia, the Ukraine and China to explore the possibility of delivering services to other nations who could benefit from citizen diplomacy efforts. The growth of the Internet and related tools present a range of opportunities to the WLS that directly and immediately affect their ability to achieve organizational objectives. The challenge for WLS today is to find and then shift resources to allow their organization to further comprehend the opportunities and to finance their development.

The emergence of the Internet and the ever rising levels of competition for funding highlight a distinct need for WLS to seek charitable donations. It has become important for a charitable organization such as WSL to have a web site that provides all relevant information to potential donors. It is also important that the site provide current, relevant information in an easily navigable format to those who seek it. It is also important that the WLS web site is frequently updated depending on the completion of their assignments. Both creation and maintenance of web sites is expensive and requires more time, especially for those who do not know how to build or how to maintain the web site. This project was begun for that purpose.

2. The Choice of Software Development Life Cycle for Web Application Tool

Software application development, commonly known as the Software Development Life Cycle, encompasses all activities to develop an application system until it is put into production. These activities include requirements gathering, analysis, design, construction, implementation, and maintenance stages. There are a wide variety of software life cycle models reported in the literature [1]. Examples for life cycle models include waterfall, iterative, rapid, spiral, RAD, Xtreme and many more. There is no single life cycle model that is best suited for every project. Different software life cycles may complement each other on the same project during different stages of development of a product. Successful use of a particular life cycle model does not ensure success or a best fit for a new project. Choosing the most appropriate model for a particular software project is challenging. Many factors such as team size, project size, available resources, deadlines or milestones, team skills and experience, business policies, and application domain may influence the choice of a life cycle model.

In the beginning phase of this project, the developer adopted the “waterfall model”, which describes a set of stages of software development in a strictly sequential manner. There are several variations of the waterfall model, but they all contain the following common stages: *requirements analysis and definition, design, implementation, testing and integration, and maintenance* [1, 2].

Soon after the requirements gathering and analysis phase started, the developer realized that it was hard to capture all the requirements in the beginning. Since it would drastically affect the design and implementation phases later, the developer, with the help from her adviser, decided to create some sample templates. The idea is that anyone should be able to use these templates to create a web site without much learning effort. However, since the customers wanted software which is easy to use the *rapid prototyping* or *throw-away prototyping* model was chosen for building the tool. During the requirements phase, a

quick-and-dirty throwaway prototype was constructed and given to potential users. This prototype was used to understand how a template works in the software. After showing the demo of templates, it became easy to (i) determine the feasibility of requirements, (ii) validate that particular functionalities were really necessary, (iii) uncover missing requirements, and (iv) determine the viability of a user interface.

Armed with the experience of using the prototype, the developer was able to complete the requirements document with increased assurance that the right system was being specified [2]. Two specific requirements were of particular interest in the initial phases:

1. Easy-to-use Graphical User Interface
2. Reliance on a blank template for the purposes of flexibility

The first requirement was addressed by making the GUI for the tool similar to that of Microsoft Word. The reason for this selection is the assumption that many PC users are familiar with Microsoft Word. For the same reason the functionalities of the tool were also chosen to mimic those of Microsoft Word. The second requirement resulted because the sponsor insisted on having one blank template instead of several other templates each for a different purposes. The sponsor was not comfortable using the other templates and hence the final implementation was left with one flexible template that can be tailored for each scenario the user wants.

The incremental prototype was created with a user interactive GUI and a blank template, which was compatible with text, fonts and images. After showing the prototype to the sponsors, they were able to create a web site easily, by trying the prototype once. And sponsor was satisfied with the blank template itself. Once the demo of the prototype was completed and the sponsor gave feedback, the requirements for the entire product were developed using a more formal SRS document. The SRS is a valuable communication tool that provided input into other phases, such as design and testing.

The following are some of the benefits of using a well defined requirements document and incremental prototyping realized by the developer for the Web Application Tool (WAT) project:

- WAT has a proven framework.
    - It has consistency and uniformity in methods and functions
    - It provides dissemination of results/deliverables in time.
- WAT facilitates information exchange.
- WAT defines and focuses roles and responsibilities.
- WAT has a predefined level of precision to facilitate a complete, correct and predictable solution.
- WAT enforces planning and control.

3. Development of The Web Application Tool

The web application tool was constructed assuming that the users may not have sufficient HTML language background but would be familiar with MS-Word. Therefore, the tool was designed in such a way that most common features are similar to those of MS-Word, but the tool enables the user to create websites rather than documents. Ideally there would be no training requirement in the use of Web Application Tool; familiarity of MS-Word should be more than enough to use this tool. Additional reasons for minimizing training are as follows:

1) There is no budget and time for such training.
2) A training requirement will act as a barrier to use due to lack of computer language knowledge, time, and motivation.

Considering most users are not actively involved in the web development, it seems unlikely such users would perceive a benefit from undertaking the training. It is also recognized that all users will have minimal knowledge of using Microsoft Word styles. Complying with the desire to create a system that involves minimal user training requires real-time processing where the user manually nominates the specific locations where the document view will be change into web page view, so that the output is WYSIWYG (‘what you see is what you get’).

3.1. Collecting requirements

The sponsor served as the application domain resource for answering various questions during the requirements gathering phase. After several meetings and interviews with the sponsors about project, the developer was able to extract initial requirements.

During requirements gathering process, several requirements were discovered that had cross-functional implications unknown to users and therfore missed. From the initial set

of requirements, the first prototypes was made. This helped in determining whether the stated requirements were unclear, incomplete, ambiguous, or contradictory; such deficiencies were resolved. The second prototype was developed after gathering most of the requirements. At this stage the requirements were prioritized because it was not feasible to implement all the requirements within the targeted deadline for the project; this decision was made jointly by the developer, advisors and sponsor.

The following are some issues recognized during the requirement phase of the Web Application Tool:

- Technical feasibility:
    - Technical feasibility is the ability of the process to take advantage of the current state of the technology in pursuing further improvement. The technical capabilities of the personnel, as well as the capability of the available technology, were considered to fulfill the requirements. But the sponsors were of little help as they did not know what is technically feasible.
- Changes in requirements:
    - As with any software development process based on prototyping approach, the clients introduced several additions and changes to the initial set of requirements when they saw the prototype.
- Software knowedge:
    - The clients' knowledge of software was tacit (the sponsors were often not able to accurately describe what they wanted).
- Uncertainty:
    - It was almost impossible to say with certainty what the actual requirements were, let alone whether or not they are met, until the system is actually in place and running. As there were lot of requirements, it was hard to say whether it is possible to fulfill all the requirements within the targeted time limit.

3.2 Design in brief

One of the biggest challenges of the project was to determine the best technologies for the project. There were numerous combinations available. During rapid prototyping, it was found that the software was going to be a substantial project and would be difficult to finish within time limit. As a result, it was very important to choose a programming language in which the developer is more comfortable, experienced. Convenient access to online documentation would also be advantageous. Fortunately, there is a web site www.mozilla.org which provides Mozilla Composer code that relies on the Gecko layout engine for rendering HTML. This was very useful for developing the Web Application Tool. But it was found that the code available on the Mozilla Composer site was written in C++. Also the code was impractical to convert into some other language. So it was decided to develop the software using Microsoft Visual C++ 6.0. However, a brief study was done to verify that the selected technology could accommodate the requirements specified by the users for the Web Application Tool.

3.3 Implementation

The implementation of WAT was divided into three phases:

1. Study Mozilla's Gecko Engine
2. Use Gecko Engine
3. Writing code to create the required tool

In order to utilize Mozilla engine, several activities were conducted to (i) understand the existing Mozilla engine in detail including its layout and working flow, and (ii) to understand it from an architectural, reuse perspective.

Useful features of using Gecko Engine are as follows:

- Gecko's function is to read web content, such as HTML, CSS, XUL, and JavaScript, and render it on user's screen.

- Gecko offers the ability to parse various document types (HTML, XML, SVG, etc), advanced rendering capabilities including composition and transformations, and support for embedded JavaScript and plug-ins.
- Gecko is a premier cross-platform, full-featured, mature and well-tested layout engine available, providing robust and high quality support for a massive array of standards. It has an ability to render web content correctly.
- Gecko expands the range of people who can actually create useful applications. It is possible to use the same technology that is used for creating web pages, such as CSS and JavaScript, in conjunction with XUL to create an application.

Challenges of using Gecko Engine:

- Gecko is large and complex, as it has complicated structure with more than 2000 files.
- This engine requires Cygwin for compilation, so it becomes necessary to compile the whole project by using Cygwin.

The GUI for the Web Application Tool was created on top of the Gecko engine. WAT GUI includes *Normal View* in the application where user can input text, images, links, tables etc. and WAT adds the HTML tags such as BODY, TABLE, IMAGE, LINK, BIG, SMALL etc. as per editing made by the user. After setting all HTML Tags WAT calls the methods from the Gecko engine to parse the document. The Gecko engine generates the HTML code using HTML tag,s text, images etc. and WAT renders the HTML code on the source screen and the web site on the preview screen.

4. Web Application Tool

This section explains the high-level architectural design of Web Application Tool, including text module, image module, paragraph module, form module, hyperlink module, table module, upload module and structure of file management design, along with some detailed usability issues that were factors in its construction.

4.1 High-Level Architectural Design

4.1.1 Process of WAT

WAT requires the data (text, images, tables etc.) to be included for creating a web site. The user can input the required data in the Normal view of WAT. WAT editor takes that data and embeds the proper HTML tags in it. With those tags WAT uses the Gecko engine to converts the data into HTML code. After the HTML code has been created, WAT displays that HTML code in the source view and stores that code as HTML file with its look and feel displayed in the preview view.

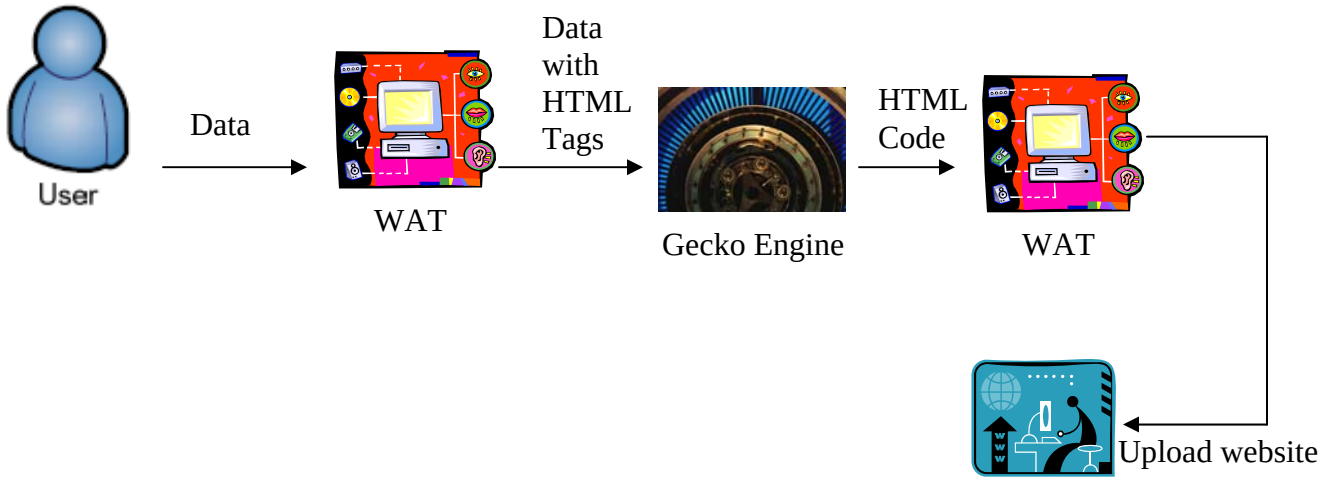


Figure 4.1 Process of Web Application Tool

4.1.2 Different view screens of WAT

Web Application Tool provides different views for accepting, modifying and displaying the data entered by user. The different view types are:

1) Normal
2) HTML Tags
3) Source
4) Preview

- *Normal* view allows a user to insert data like texts, images, hyperlinks, tables … etc. to make web pages.
- *HTML Tags* view allows a user to view the HTML tags available to create web pages.
- *Source* view allows a user to view and edit the HTML code generated by the tool based on the data entered by the user in normal screen.
- Lastly, *Preview,* as the name suggests, allows a user to see how a web page will look like.

4.1.3 Different Modules of WAT

The high level architecture breaks down for Accepting, Modifying, and Uploading the user data as html pages, is given as different modules mentioned below -

Text module: This module allows user to add, delete and modify the text entered by the user in the normal view of WAT. This also allows user to change the font, size, style, alignment including color of the text.

Image module: This module allows user to add, delete and resize image selected by the user. It also allows the user to change the appearance like spacing between image and text, apply solid border for an image and aligning text to image.

Paragraph module: This module allows user to edit setting for text, like paragraph, different types of headings and bulleted and numbering list.

Form module: This module allows the user to insert the things required like button, text field, text area, check box etc. for creating desirable form.

Hyperlink module: This module allows user to set the link for the user as per user's specification for the selected text or an image.

Table module: This module allows user to add, modify and delete table with insert cell/row/column, delete cell/row/column, split and merge inside the table.

Upload module: This module allows user to upload the contents which s/he can see in preview, after providing web site name, ftp address, username and password.

4.2 Text module

When user enters some text in the ‘Normal View’ provided by WAT, default values are assigned for text properties like font, size, color etc. WAT also allows user to edit the text properties by changing font, color, size. Once the user starts editing text properties, WAT will add HTML tags as per the editing done by the user. Then WAT will invoke the appropriate command of the Gecko engine by passing required parameters (command string, boolean flag for showing UI and value string).

e.g. to invoke the bold command - editableDocument.execCommand("Bold", false, null);

The *Source View* of WAT will display the HTML code generated by Gecko engine for the contents present in ‘Normal View’. Following are some of the examples given for commands use by the WAT:

command	value	explanation / behavior
backcolor	?	This command will set the background color of the document.
bold	none	If there is no selection, the insertion point will set bold for subsequently typed characters. If there is a selection and all of the characters are already bold, the bold will be removed. Otherwise, all selected characters will become bold.
copy	none	If there is a selection, this command will copy the selection to the clipboard. If there isn't a selection, nothing will happen.
cut	none	If there is a selection, this command will copy the selection to the clipboard and remove the selection from the edit control. If there isn't a selection, nothing will happen.
decreasefontsize	none	This command will add a <small> tag around selection or at insertion point.
delete	none	This command will delete all text and objects that are selected.
fontname	?	This command will set the fontface for a selection or at the insertion point if there is no selection.
fontsize	?	This command will set the fontsize for a selection or at the insertion point if there is no selection.
forecolor	?	This command will set the text color of the selection or at the insertion point.
increasefontsize	none	This command will add a <big> tag around selection or at insertion

		point.
inserthtml	valid html string	This command will insert the given html into the <body> in place of the current selection or at the caret location.
italic	none	If there is no selection, the insertion point will set italic for subsequently typed characters. If there is a selection and all of the characters are already italic, the italic will be removed. Otherwise, all selected characters will become italic.
paste	none	This command will paste the contents of the clipboard at the location of the caret. If there is a selection, it will be deleted prior to the insertion of the clipboard's contents.
redo	none	This command will redo the previous undo action. If undo was not the most recent action, this command will have no effect.
selectall	none	This command will select all of the contents within the editable area.
subscript	none	If there is no selection, the insertion point will set subscript for subsequently typed characters. If there is a selection and all of the characters are already subscripted, the subscript will be removed. Otherwise, all selected characters will be drawn slightly lower than normal text.
superscript	none	If there is no selection, the insertion point will set superscript for subsequently typed characters. If there is a selection and all of the characters are already superscripted, the superscript will be removed. Otherwise, all selected characters will be drawn slightly higher than normal text
underline	none	If there is no selection, the insertion point will set underline for subsequently typed characters. If there is a selection and all of the characters are already underlined, the underline will be removed. Otherwise, all selected characters will become underlined.
undo	none	This command will undo the previous action. If no action has occurred in the document, then this command will have no effect.

Table 4.2.1 Commands used for Text editing and its behavior

? – value is depends upon selection made by the user

Figure 4.2.1 shows the WAT screen after entering and editing text. The HTML tags and HTML code for the text screens follow in Figures 4.2.2 through 4.2.4.

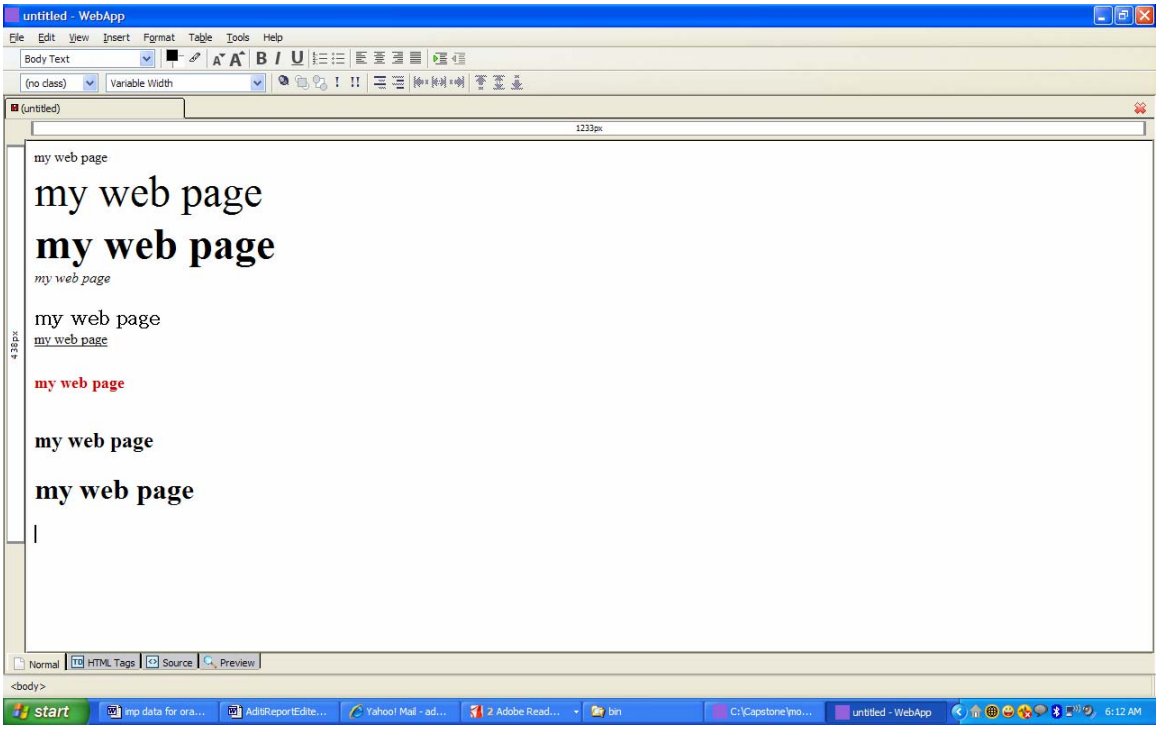


Figure 4.2.1 Normal view of WAT after entering some texts

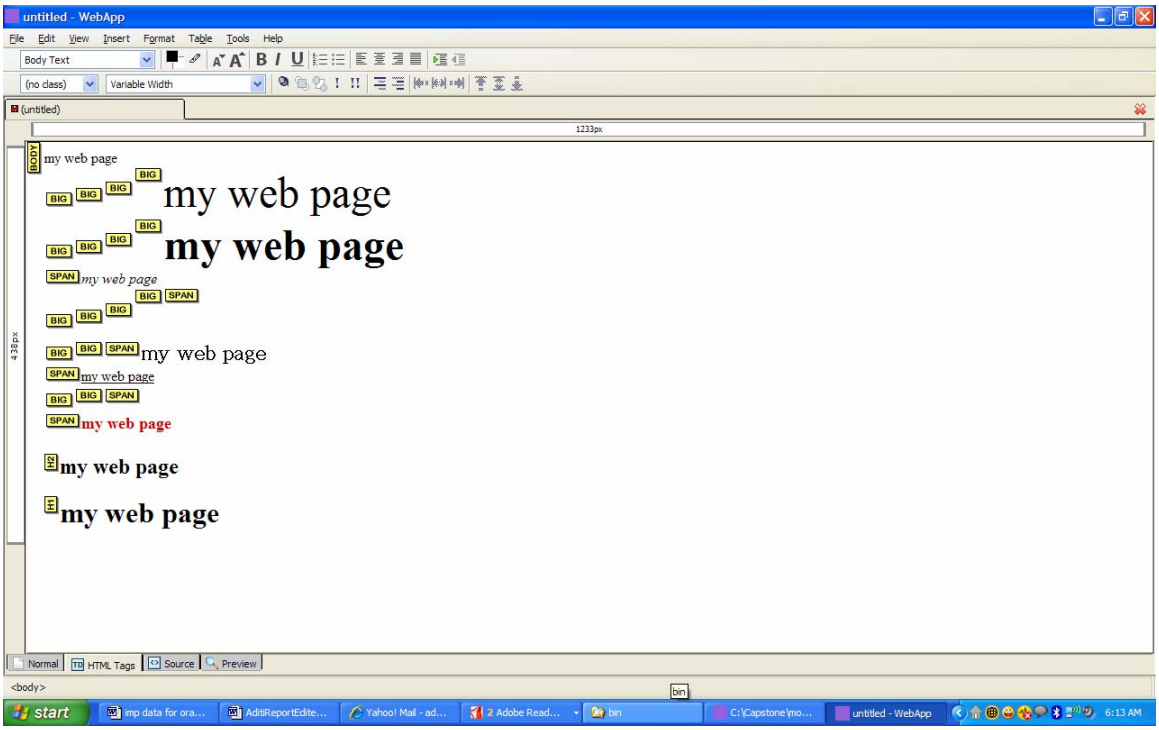


Figure 4.2.2 HTML Tag view of WAT for figure 4.2.1

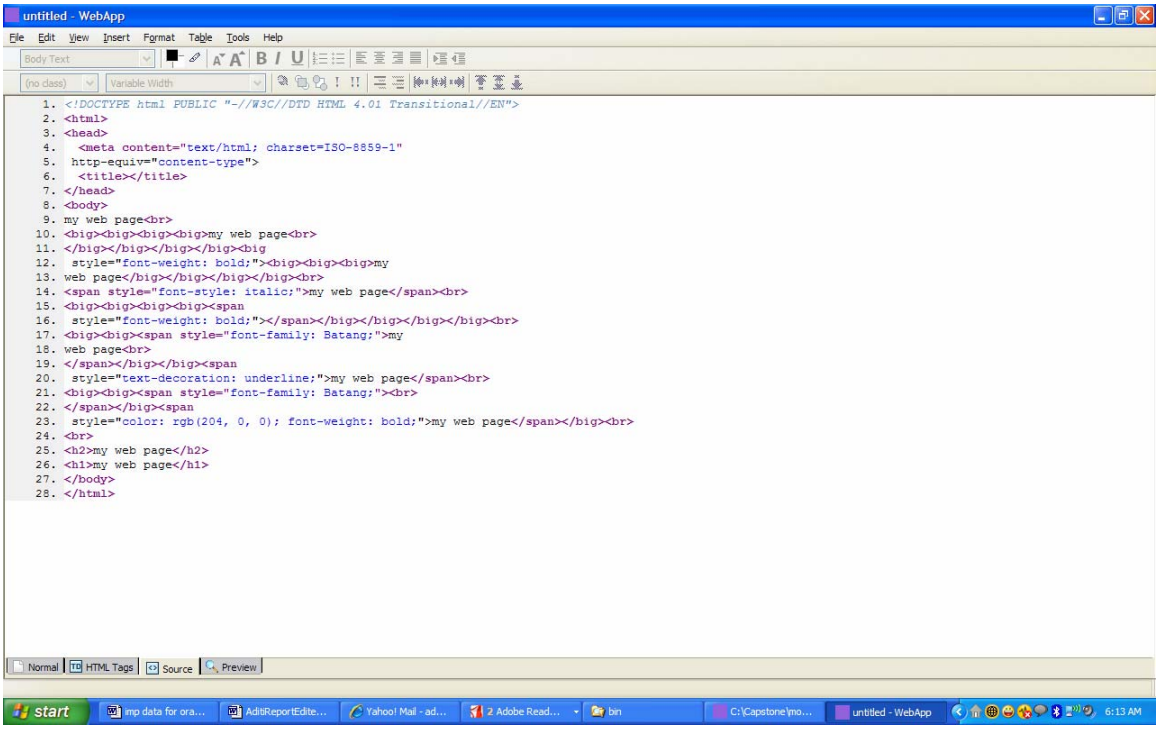


Figure 4.2.3 Source view of WAT for figure 4.2.1

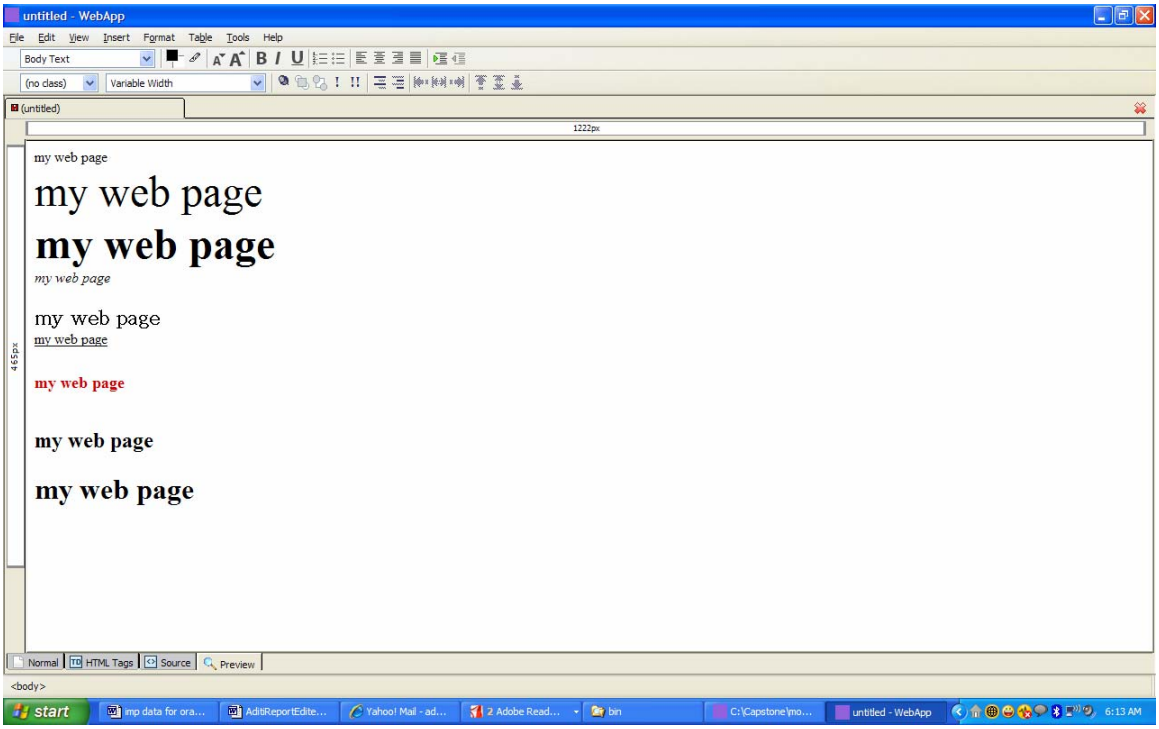


Figure 4.2.4 Preview of WAT for figure 4.2.1

4.3 image module

The User can enter an image in WAT by providing the location and other optional image properties. The optional image properties include dimensions for an image, appearance of an image, spacing from left, right, top, bottom, alignment of text to image, hyperlink for the image, and so forth. Once the user provides the specifications for an image, WAT will enable the design mode for the image and add HTML tags according to the editing done by the user. Then WAT invokes the appropriate command of the Gecko engine by passing required parameters. The *Source View* of WAT will display the HTML code generated by Gecko engine for the contents present in *Normal View*.

Following are some of the examples given for commands use by the WAT:

command	value	explanation / behavior
insertimage	URL	This command will insert an image (referenced by url) at the insertion point.
copy	none	If there is a selection, this command will copy the selection to the clipboard. If there isn't a selection, nothing will happen.
cut	none	If there is a selection, this command will copy the selection to the clipboard and remove the selection from the edit control. If there isn't a selection, nothing will happen.
delete	none	This command will delete all text and objects that are selected.
paste	none	This command will paste the contents of the clipboard at the location of the caret. If there is a selection, it will be deleted prior to the insertion of the clipboard's contents.
redo	none	This command will redo the previous undo action. If undo was not the most recent action, this command will have no effect.

Table 4.3.1 Commands used for Image editing and its behavior

Figure 4.3.1 shows the WAT screen after entering and editing images. Created the HTML tags and HTML code for the image screens follow in Figures 4.3.2 through 4.3.4.

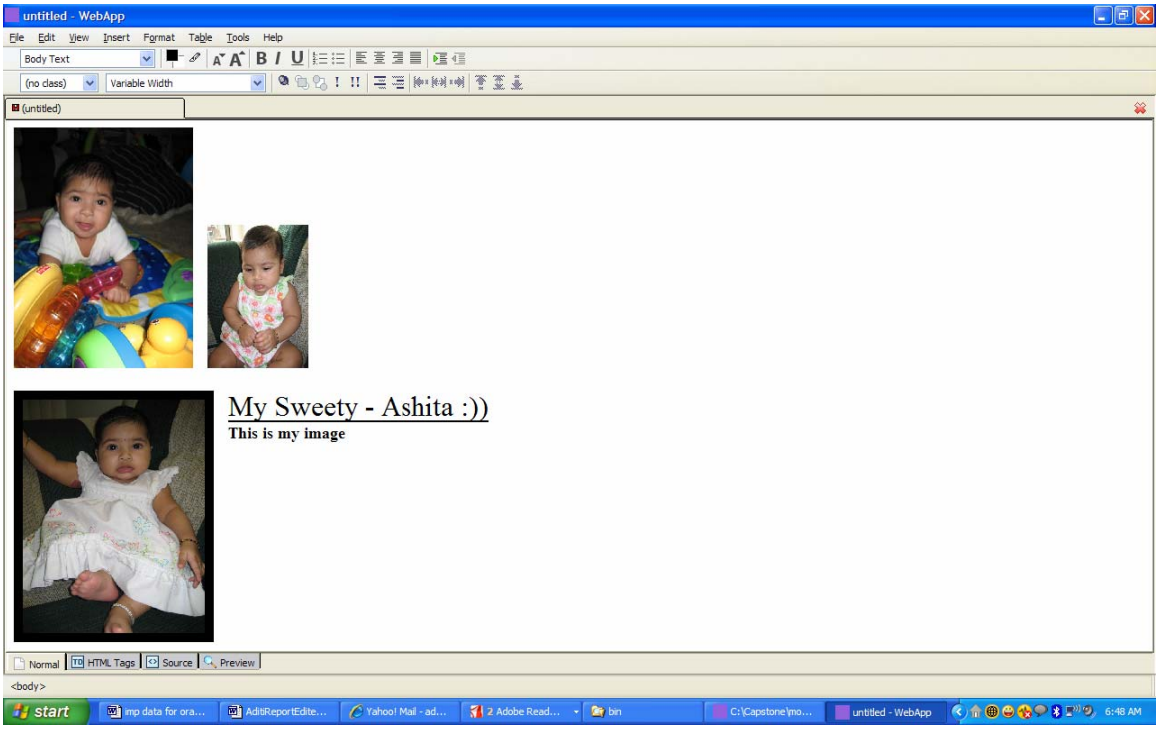


Figure 4.3.1 Normal view of WAT after entering some images

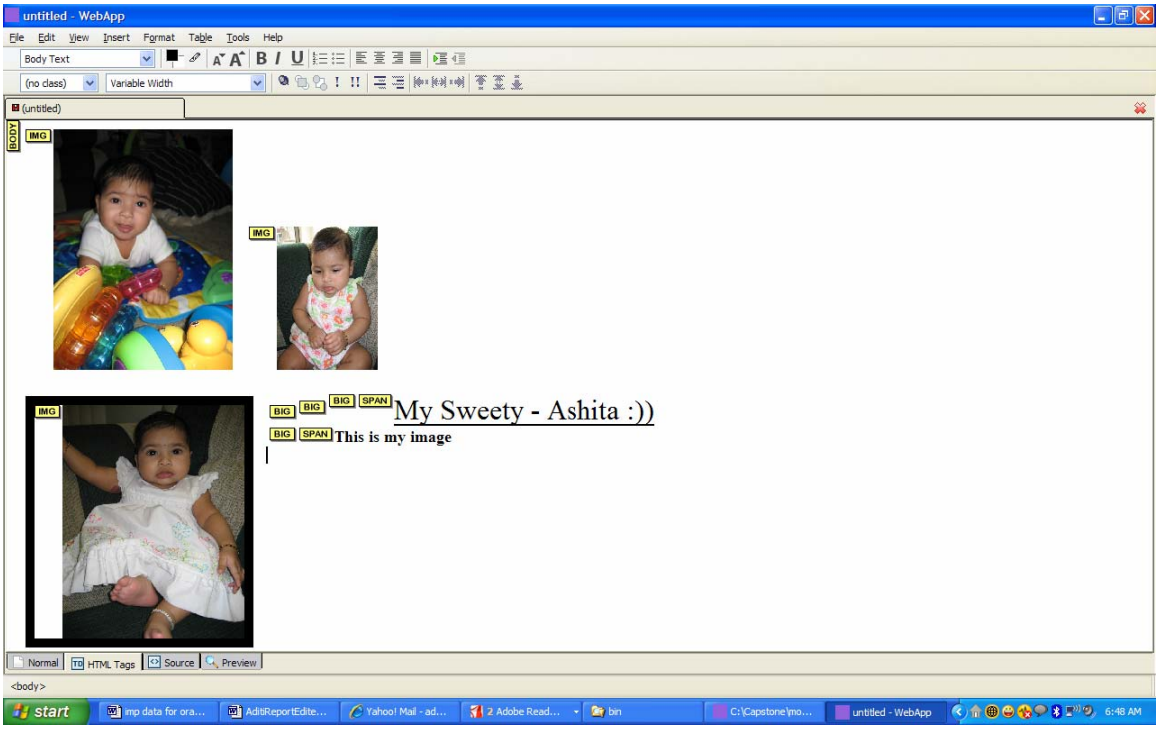


Figure 4.3.2 HTML Tag view of WAT for figure 4.3.1

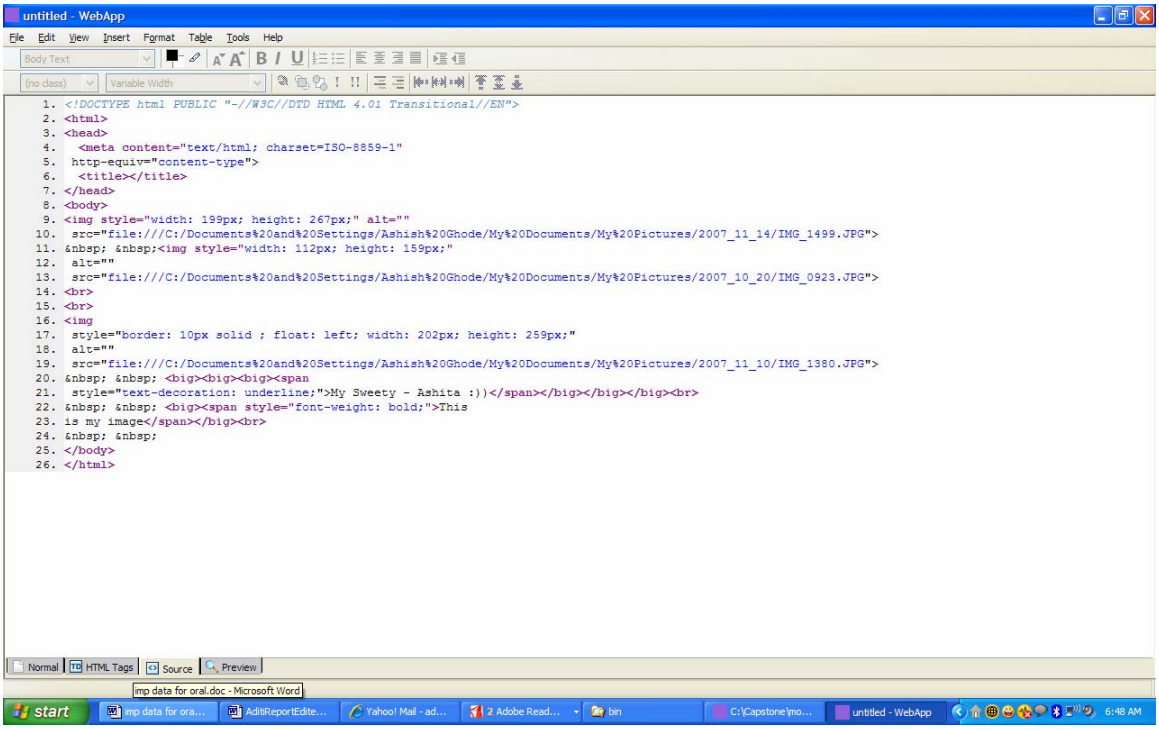


Figure 4.3.3 Source view of WAT for figure 4.3.1

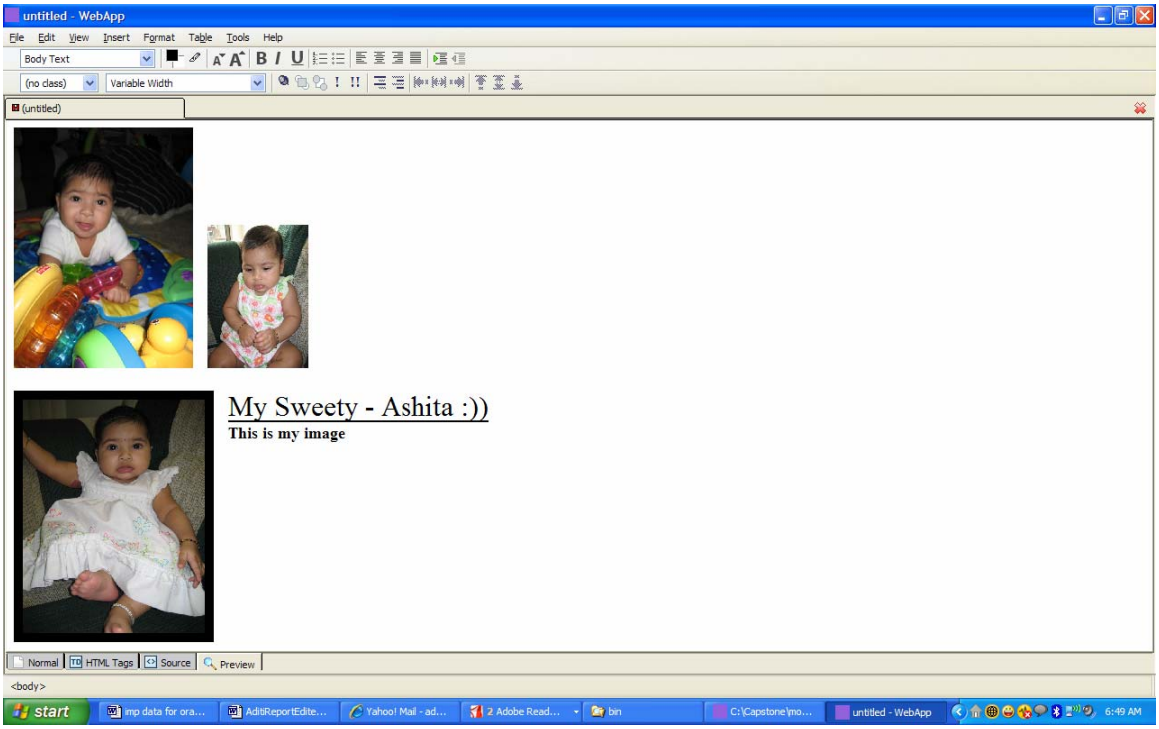


Figure 4.3.4 Preview of WAT for figure 4.3.1

4.4 Paragraph module

WAT's Paragraph module provides different text formatting settings for the selected text. The different settings available are body text, paragraph, Heading 1, Heading 2, Heading 3, Heading 4, Heading 5, Heading 6 and address. The default text format is body text. Maximum six levels of Headings are available.

As per user's selection, WAT will enable the design mode for the paragraph and add HTML tags as per the selection done by the user. Then WAT invokes the appropriate command of the Gecko engine by passing required parameters. The *Source View* of WAT will display the HTML code generated by Gecko engine for the contents present in *Normal View*.

Examples of heading levels:

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Figure 4.4.1 shows the WAT screen after entering and editing text using paragraph. Created the HTML tags and HTML code for the text screens follow in Figures 4.4.2 through 4.4.4.

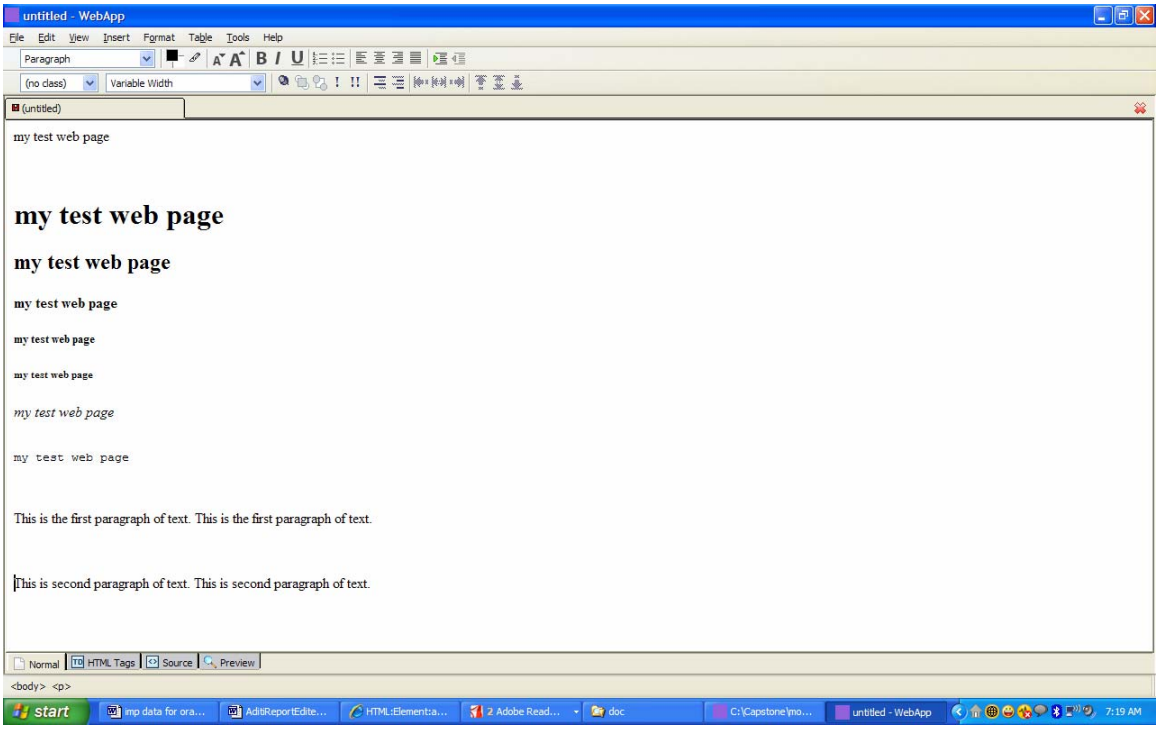


Figure 4.4.1 Normal view of WAT after using paragraph

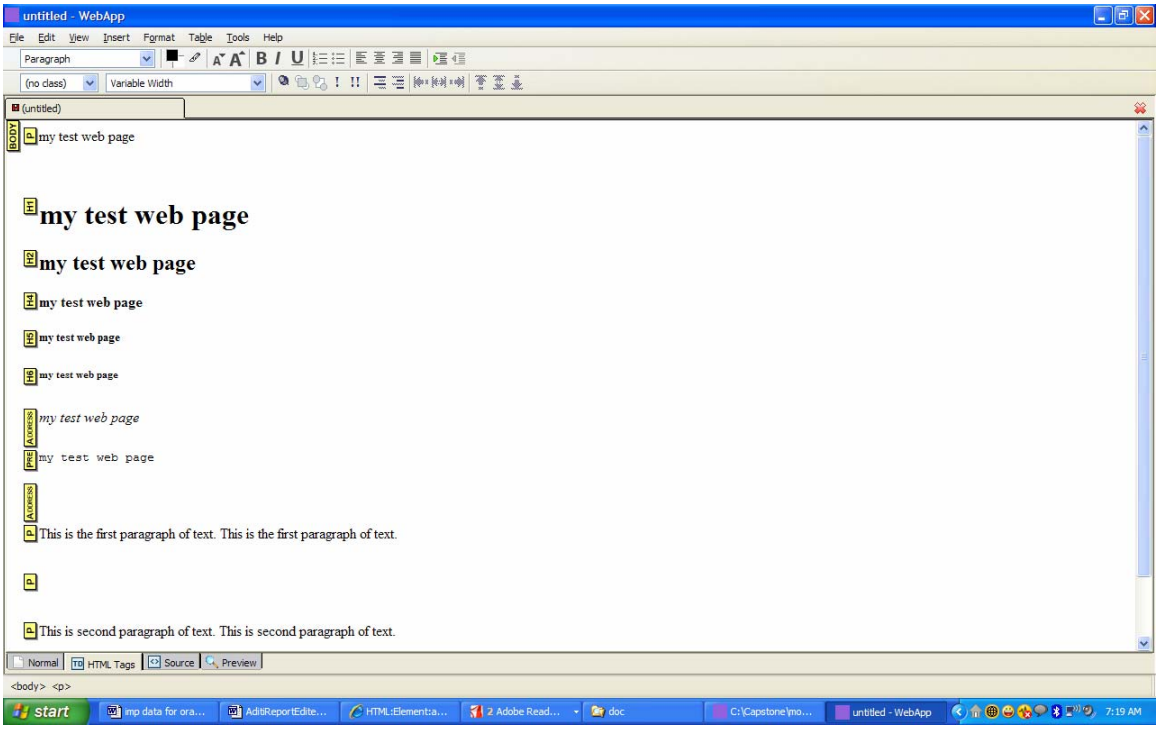


Figure 4.4.2 HTML Tag view of WAT for figure 4.4.1

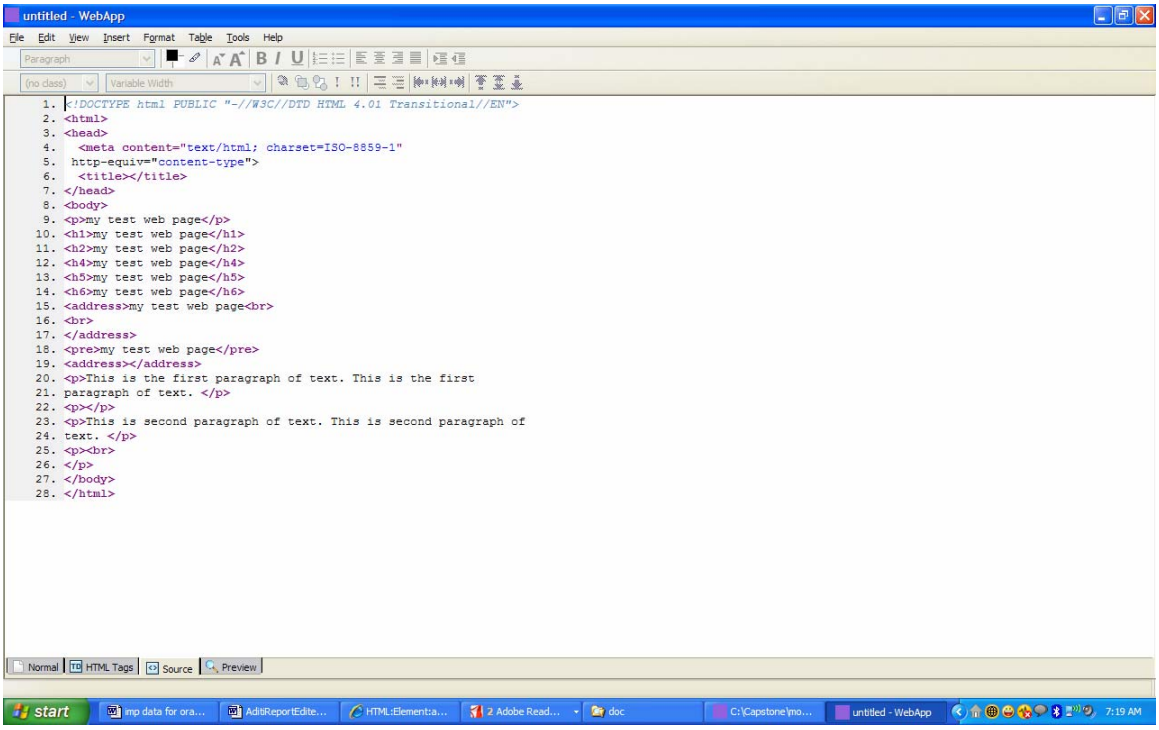


Figure 4.4.3 Source view of WAT for figure 4.4.1

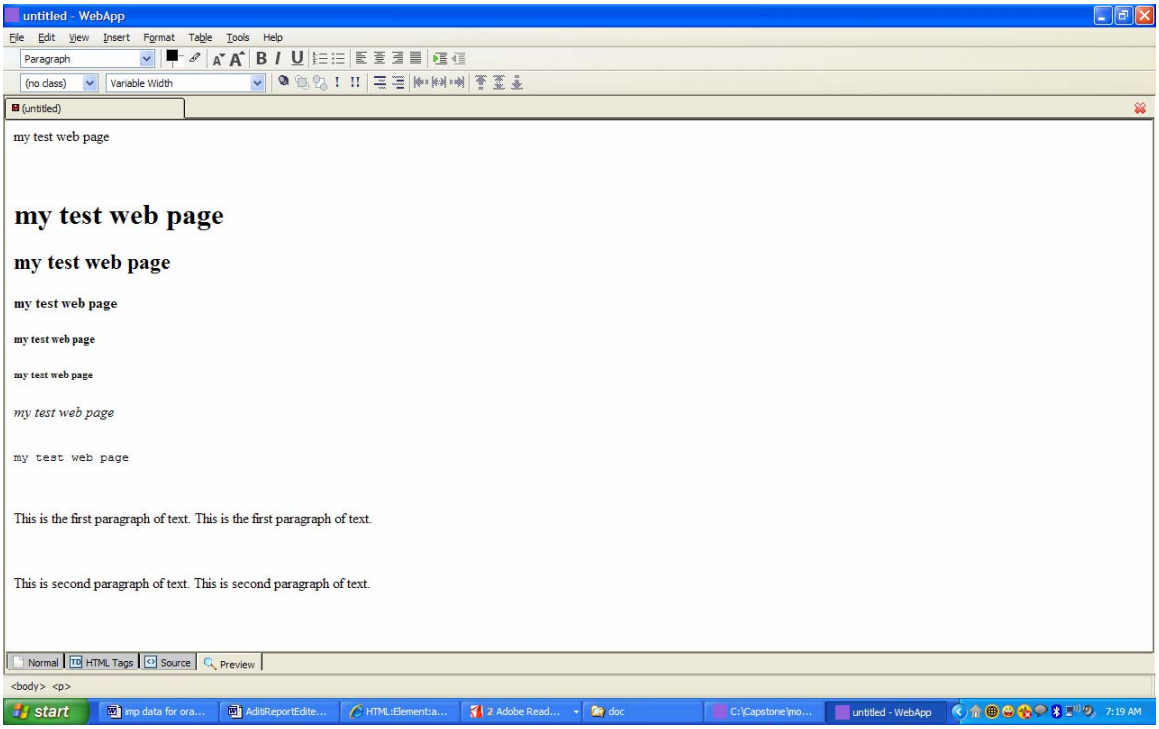


Figure 4.4.4 Preview of WAT for figure 4.4.1

4.5 Form module

This module allows user to insert different Form elements such as checkboxes, radio buttons, and selection lists. Each Form element used by the user is considered as a distinct object. WAT maintains an element array containing an entry for each Form element such as checkbox, radio button, text area etc per Form in the document. WAT assigns a HTML tag for each form and each individual element in the form entered by user. It then invokes the Gecko's java runtime engine to create a Form object for each form in the document, which uses standard HTML syntax with the addition of JavaScript event handler. WAT provides user the facility to enter a value for the NAME attribute of a Form and its elements, which WAT uses to index into the forms array.

If multiple objects on the same form have the same NAME attribute, an array of the given name is created automatically. Each element in the array represents an individual Form object. Elements are indexed in source order starting at 0. For example, if two Text elements and a TextArea element on the same form have their NAME attribute set to "myField", an array with the elements myField[0], myField[1], and myField[2] is created.

Figure 4.5.1 shows the WAT screen after entering some text using forms. Created the HTML tags and HTML code for the text screens follow in Figures 4.5.2 through 4.5.4.

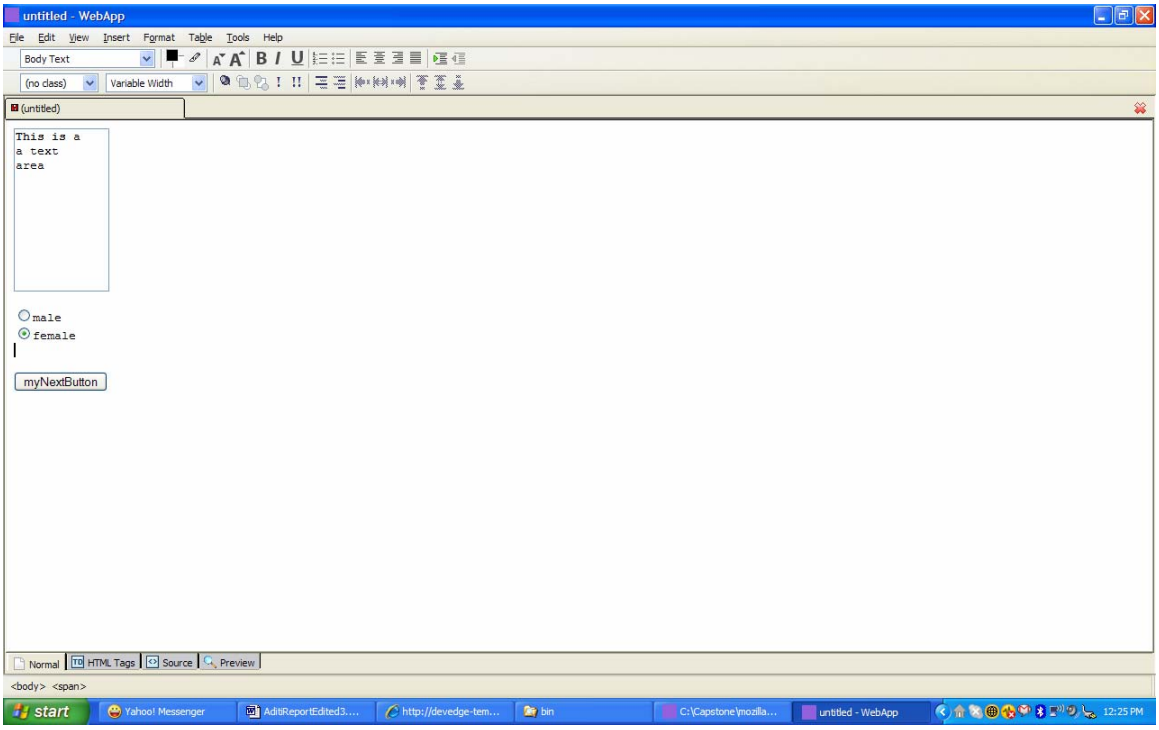


Figure 4.5.1 Normal view of WAT after using forms

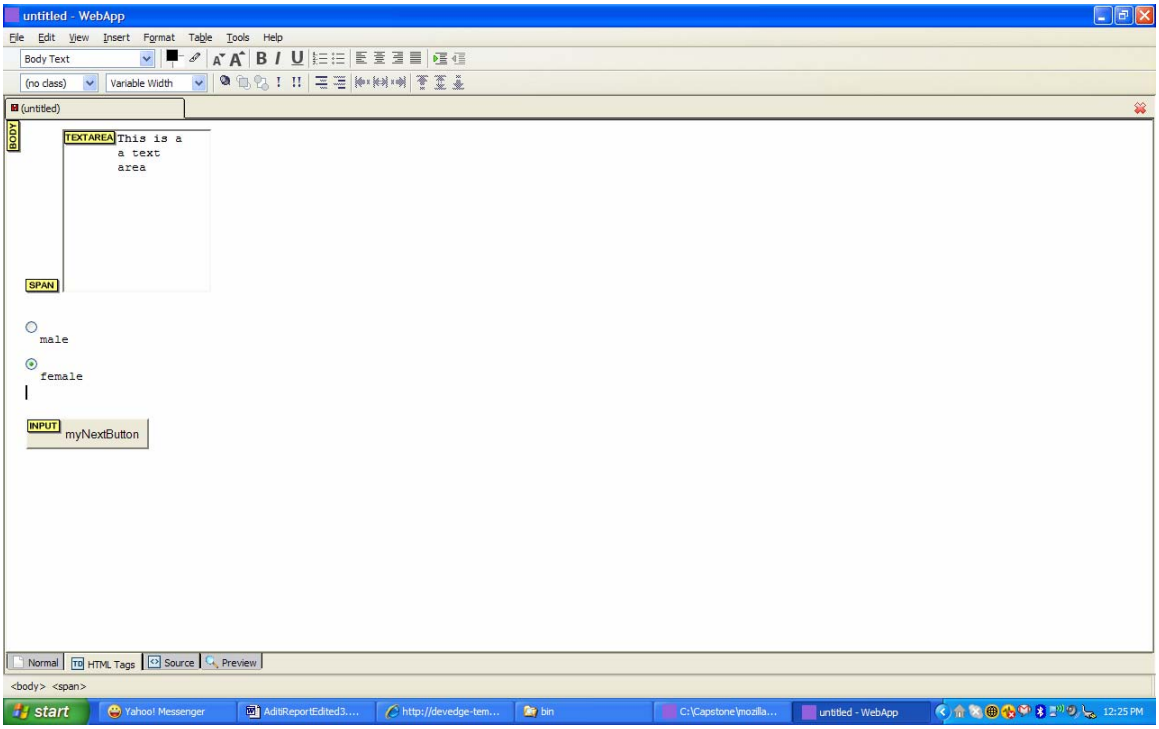


Figure 4.5.2 HTML Tag view of WAT for figure 4.5.1

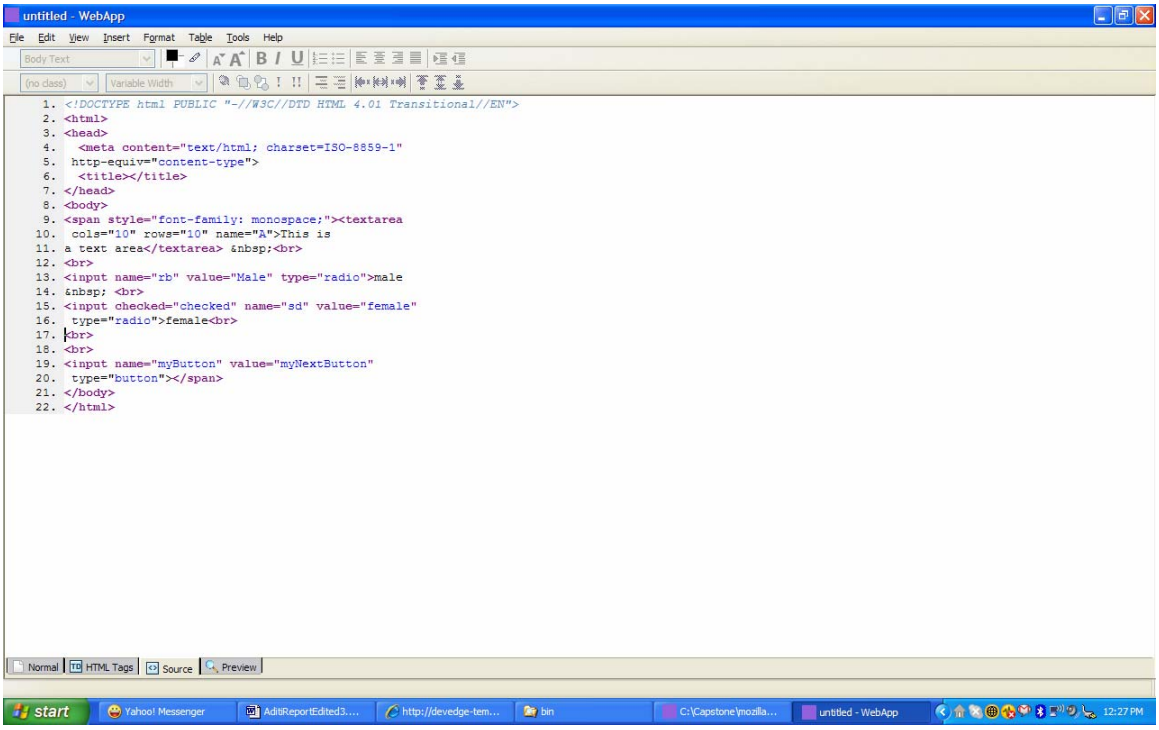


Figure 4.5.3 Source view of WAT for figure 4.5.1

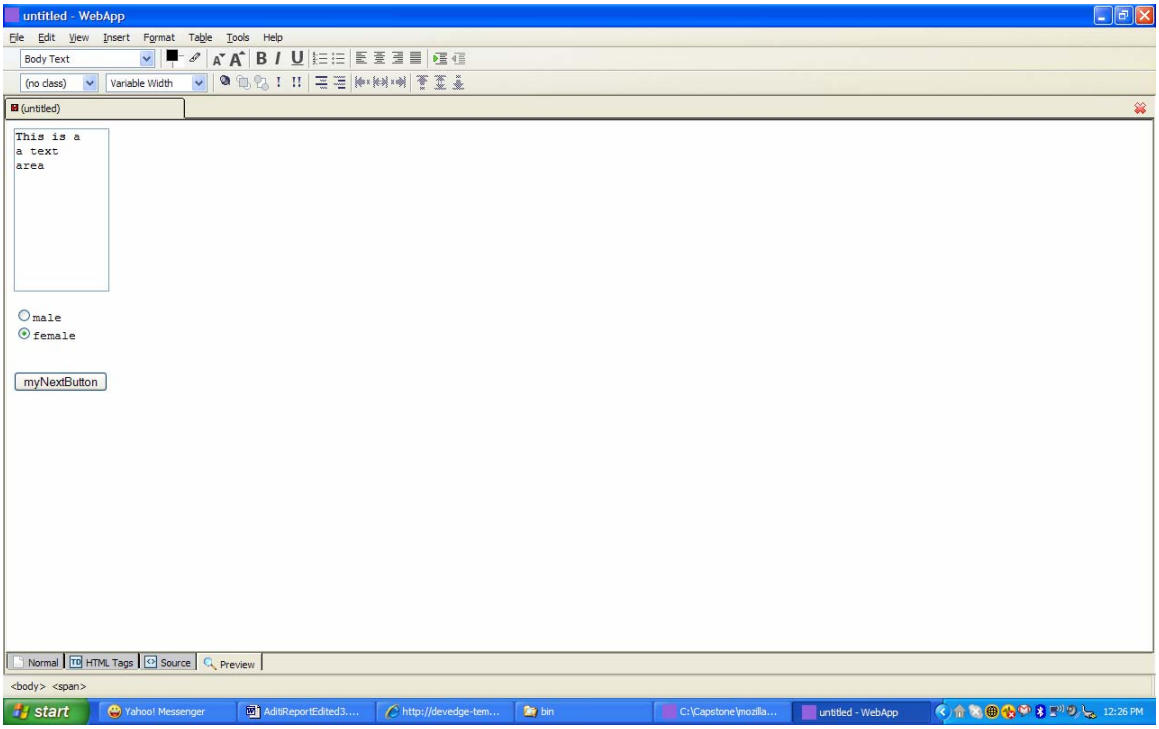


Figure 4.5.4 Preview of WAT for figure 4.5.1

4.6 Hyperlink module

This module deals with the link provided for a text or an image. The hyperlink module allows a user to select the required text or an image and provide a link location for selected text or an image. When the user clicks the link text or image, the WAT will load the link hypertext reference into its target window by adding a link with the ‘A’ tag.

The Gecko engine create a Link object corresponding to each ‘A’ tag in the document that supplies the HREF attribute. Gecko puts these objects as an array in the *document.links* property. WAT accesses a Link object by indexing this array. WAT considers each link provided by the user for text or an image as an object. And each Link object has a location object and has the same properties as a location object.
The following is the brief description for some of the properties used:

Property	**Description**
host	Specifies the host and domain name, or IP address, of a network host.
href	Specifies the entire URL.
pathname	Specifies the URL-path portion of the URL.
port	Specifies the communications port that the server uses.
protocol	Specifies the beginning of the URL, including the colon.
search	Specifies a query string.
target	Reflects the TARGET attribute.
text	A string containing the content of the corresponding A tag.
x	The horizontal position of the link's left edge, in pixels, relative to the left edge of the document.
y	The vertical position of the link's top edge, in pixels, relative to the top edge of the document.

Table 4.6.1 Commands used for Hyper link editing and its behavior

Figure 4.6.1 shows the WAT screen for hyper links. Created the HTML tags and HTML code for the image screens follow in Figures 4.6.2 through 4.6.4.

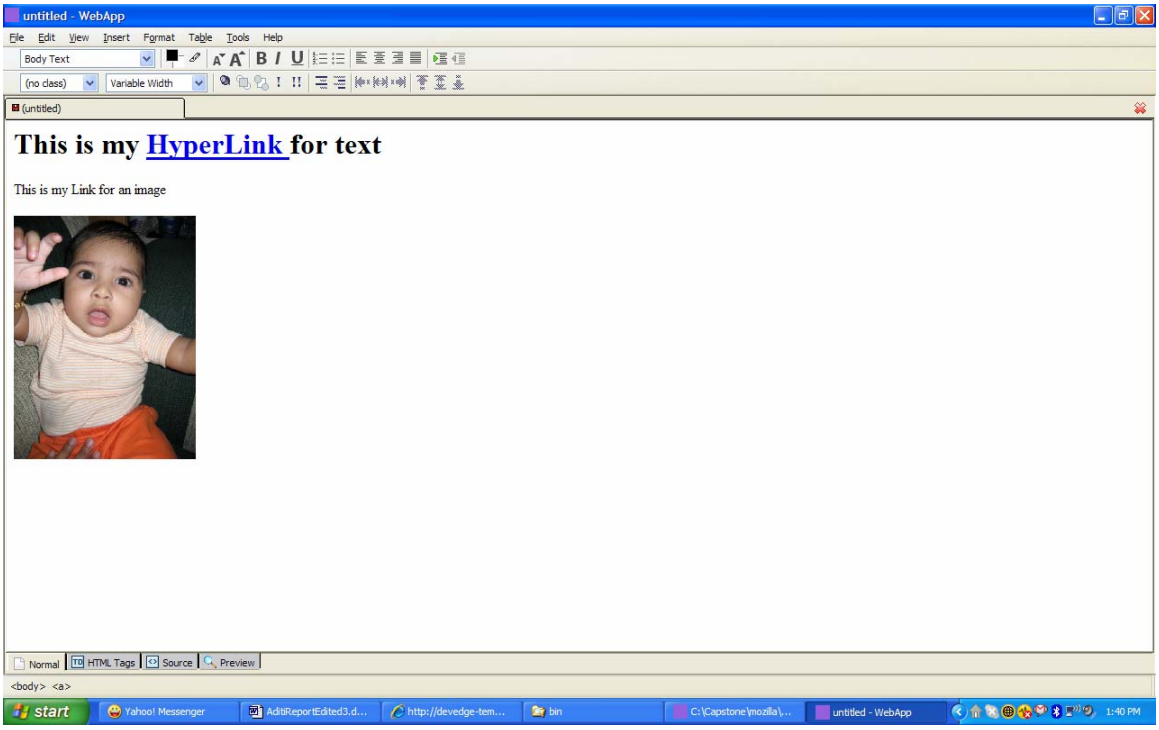


Figure 4.6.1 Normal view of WAT after using hyperlinks

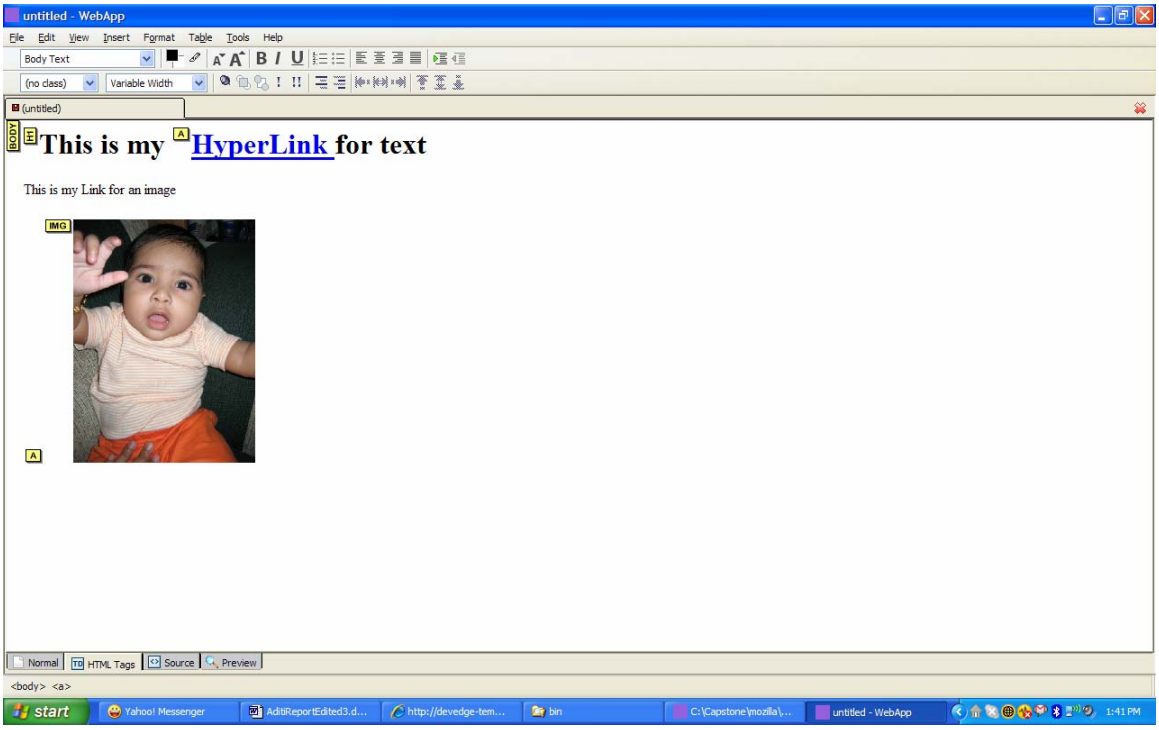


Figure 4.6.2 HTML Tag view of WAT for figure 4.6.1

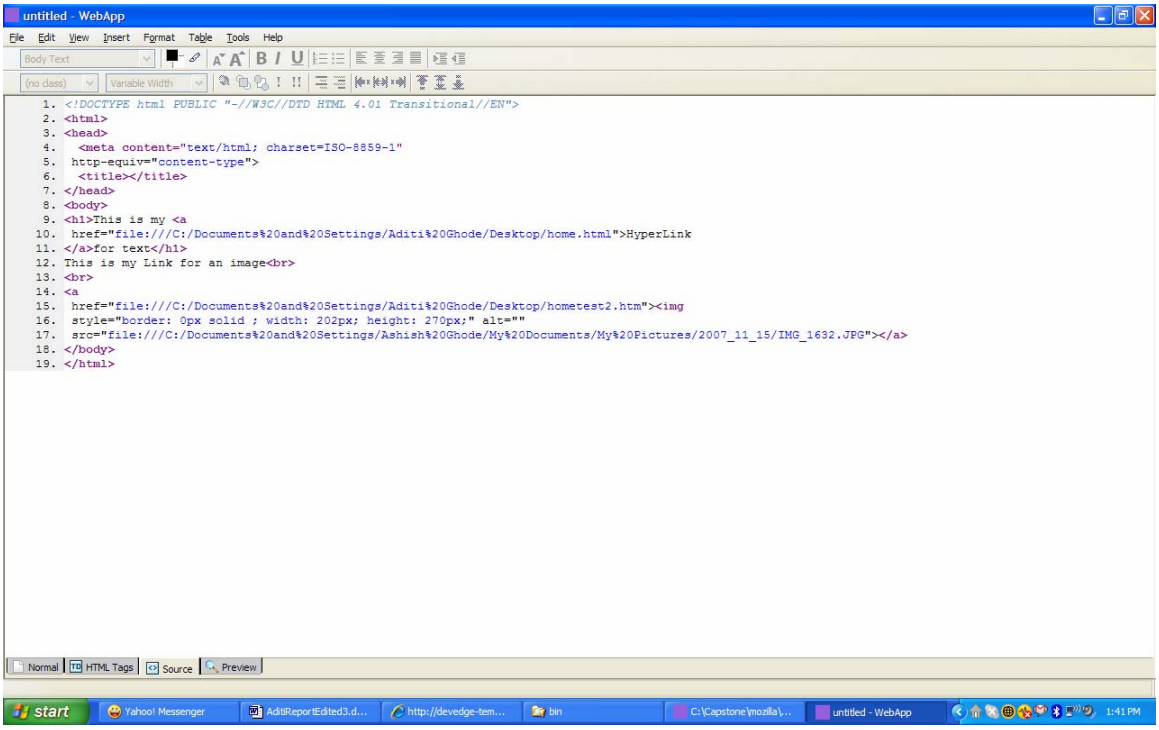


Figure 4.6.3 Source view of WAT for figure 4.6.1

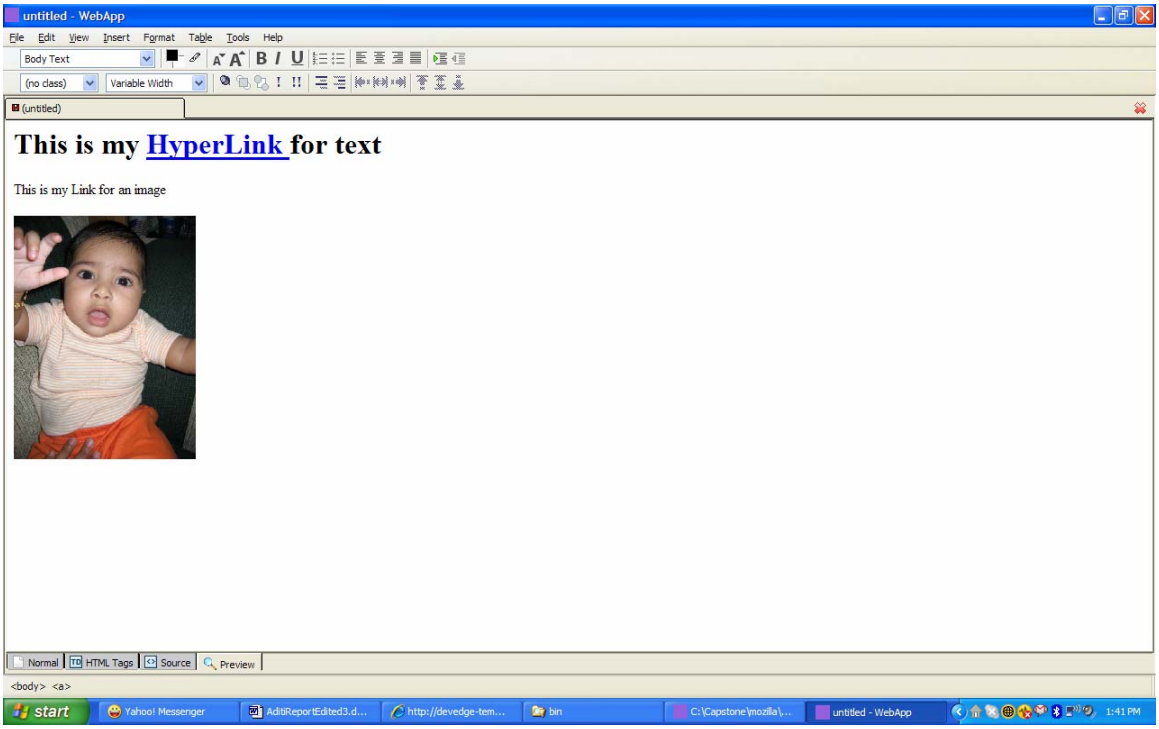


Figure 4.6.4 Preview of WAT for figure 4.6.1

4.7 Table module

This module deals with table structures in WAT. Table structures allow a user to insert, delete, modify table. Users can also change the table's width, change the table's height, insert row, insert column, delete row, delete column, insert cell and delete cell. When a user inserts the table, WAT takes the number of columns and rows from the user, and adds the HTML tags accordingly, which actually follows a tree structure and send that structure to the Gecko engine to create an HTML code.

Following is the example, when user inserts 2 X 2 table (with 2 columns and 2 rows) in the normal view. WAT follows the tree structure shown in Figure 4.7.1 below, and adds the HTML tags accordingly.

hi	there
hello	world

Table 4.7.1 Table with 2 columns and 2 rows

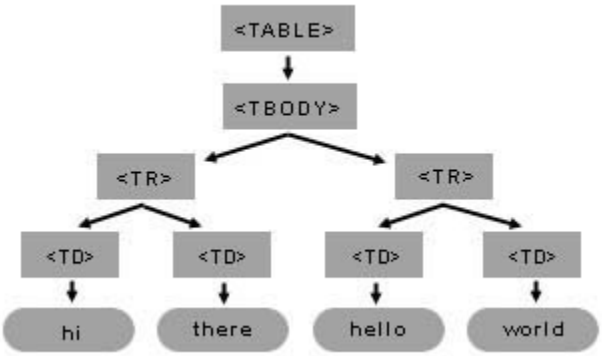


Figure 4.7.1 Tree structure for Table 4.7.1

Figure 4.7.2 shows the WAT screen after entering the table shown in Table 4.7.1. Created the HTML tags and HTML code for the text screens follow in Figures 4.7.3 through 4.7.5.

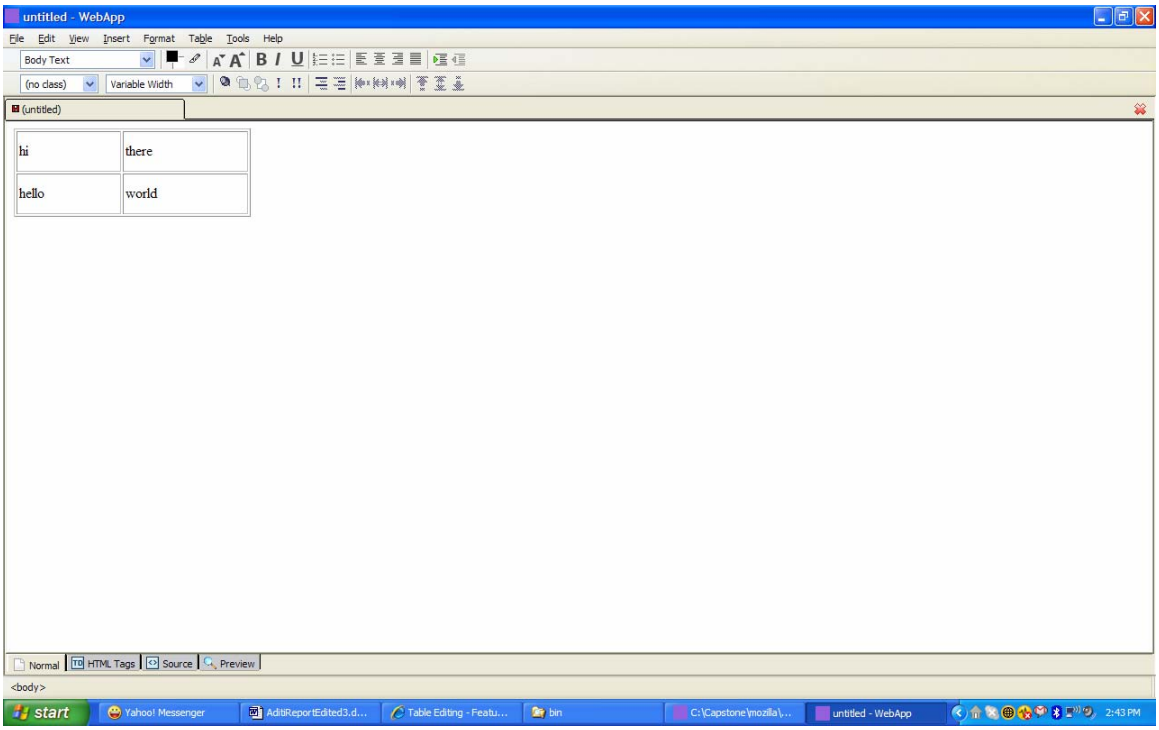


Figure 4.7.2 Normal view of WAT after inserting a 2 x 2 table

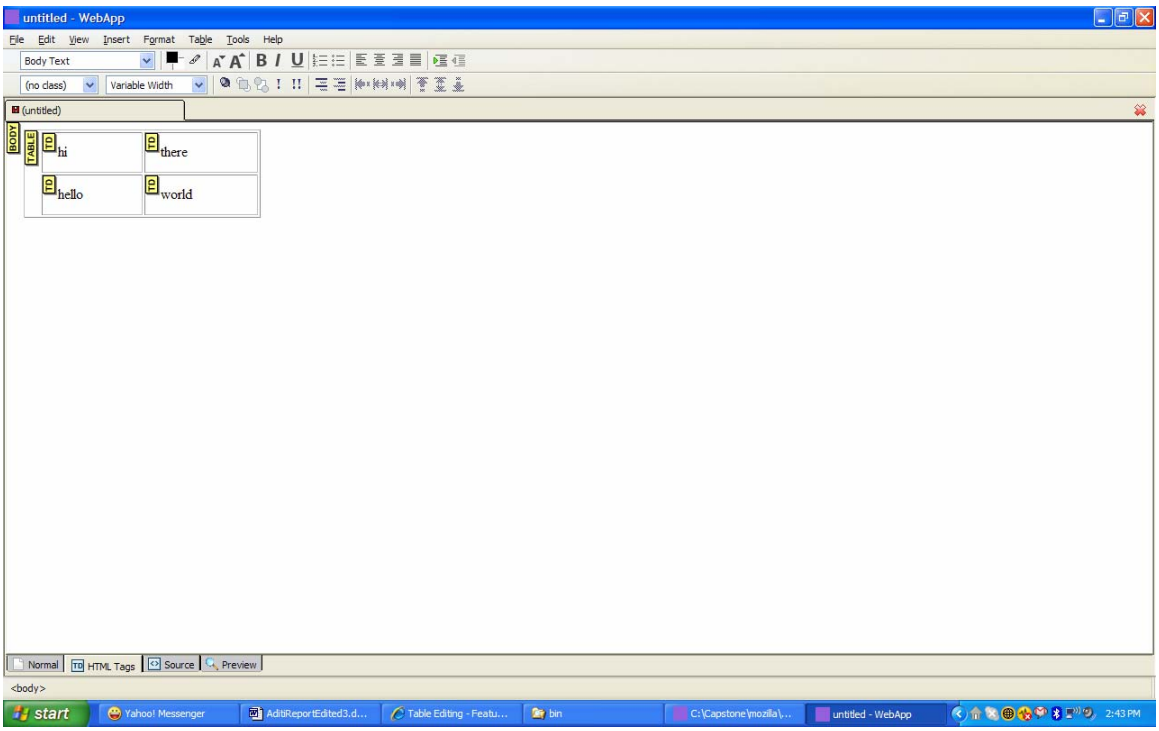


Figure 4.7.3 HTML Tag view of WAT for figure 4.7.2

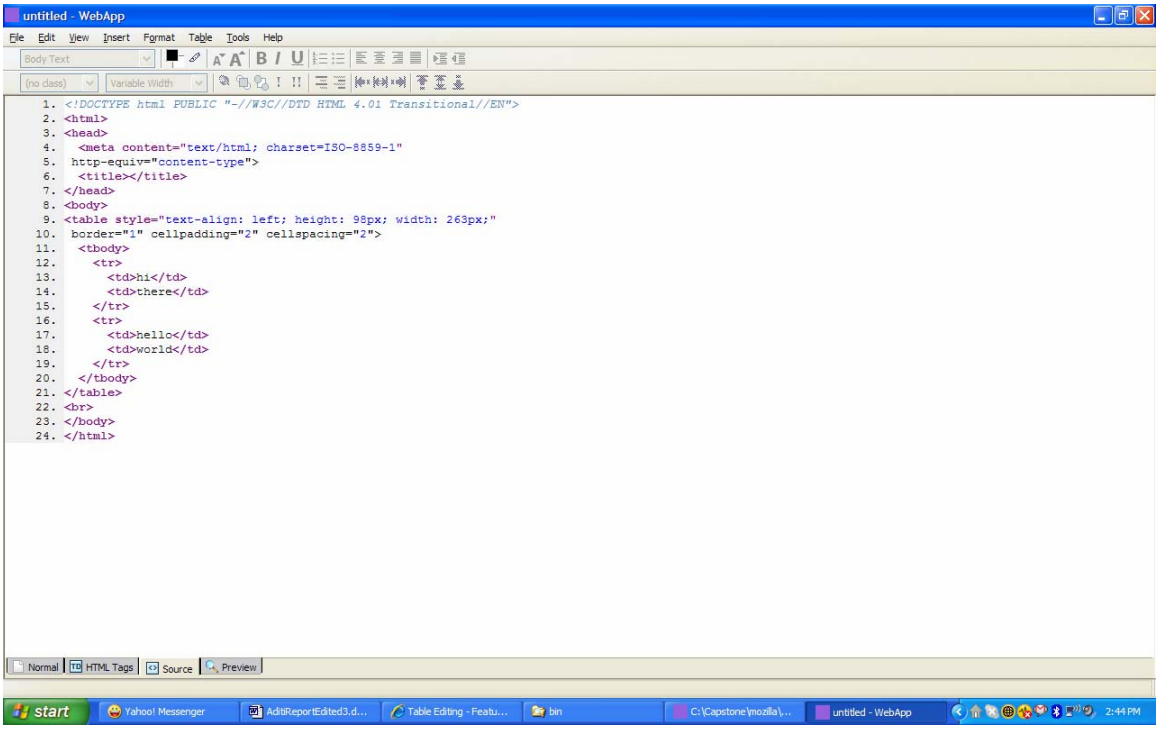


Figure 4.7.4 Source view of WAT for figure 4.7.2

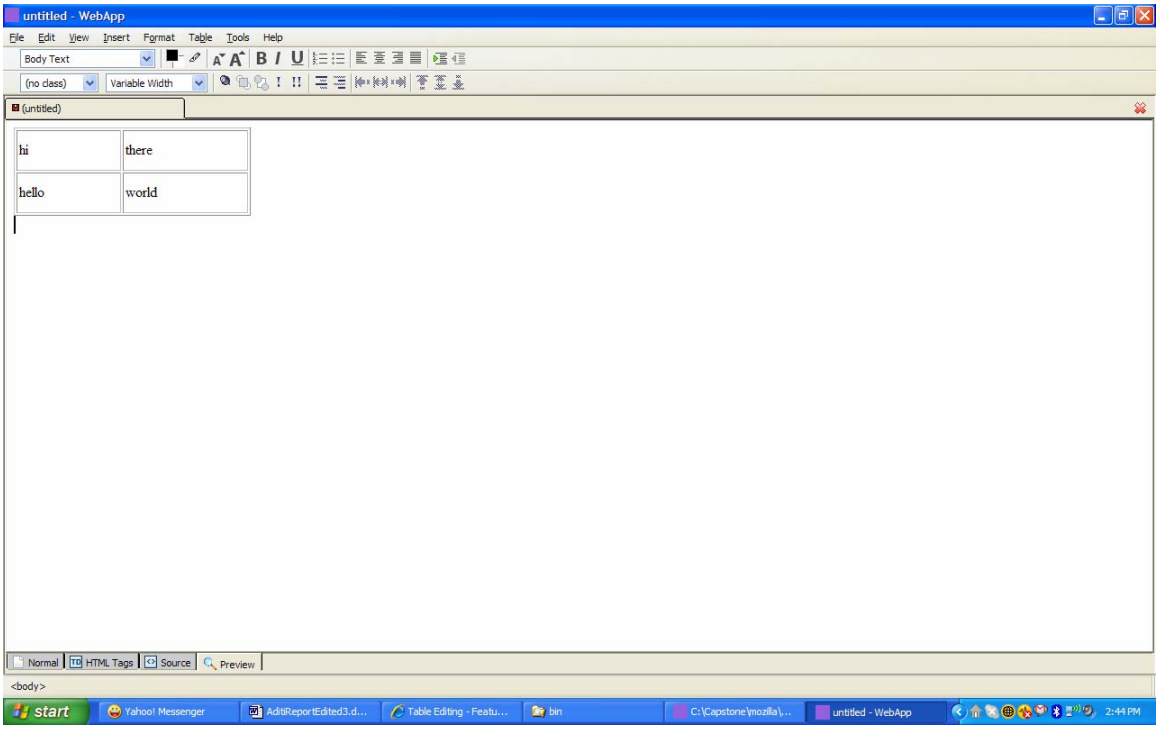


Figure 4.7.5 Preview of WAT for figure 4.7.2

4.8 Upload website module

This module provides facilities to publish or upload the web site on ftp (File Transfer Protocol) or http.

FTP is the default location from which the user wants to upload the Web pages using the File Transfer Protocol. User may need to contact their Internet service provider to find the details to be entered. For example, if the service provider is America Online, the URL in this box might look something like: ftp://ftp.aol.com/docs/username/index.html.

HTTP is the default location to which the user wants to upload the Web pages using the Web server protocol. If users are not running their own Web server, they may need to contact their internet service provider to get the required details such as the URL, which will look like http://commercialweb.com/docs/username/index.html.

For publishing the web site the user must provide WAT information such as site name, web site information, publishing server information with username and password. Wat will copy and paste all the web pages, creating a folder for other files especially for images onto specified server.

Figures 4.8.1 through 4.8.4 show the WAT screen which allows user to publish their web site followed by the screens while web site is publishing and after the publish process complete.

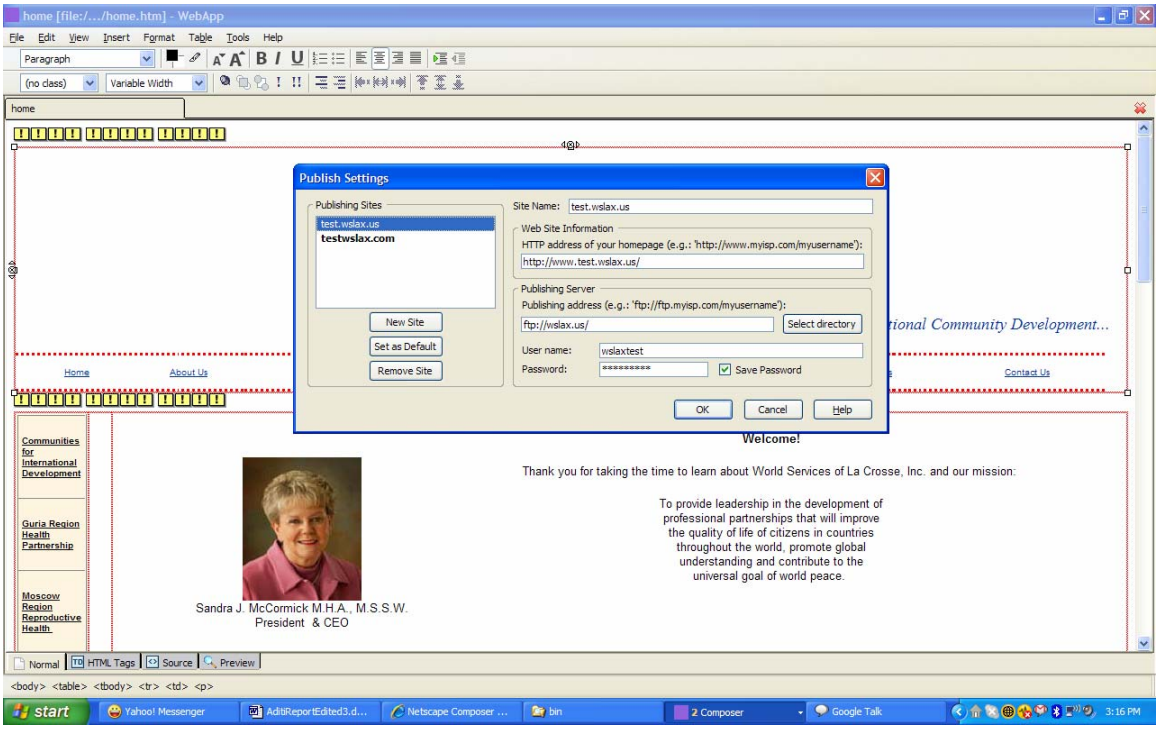


Figure 4.8.1 Normal view of WAT for publish settings

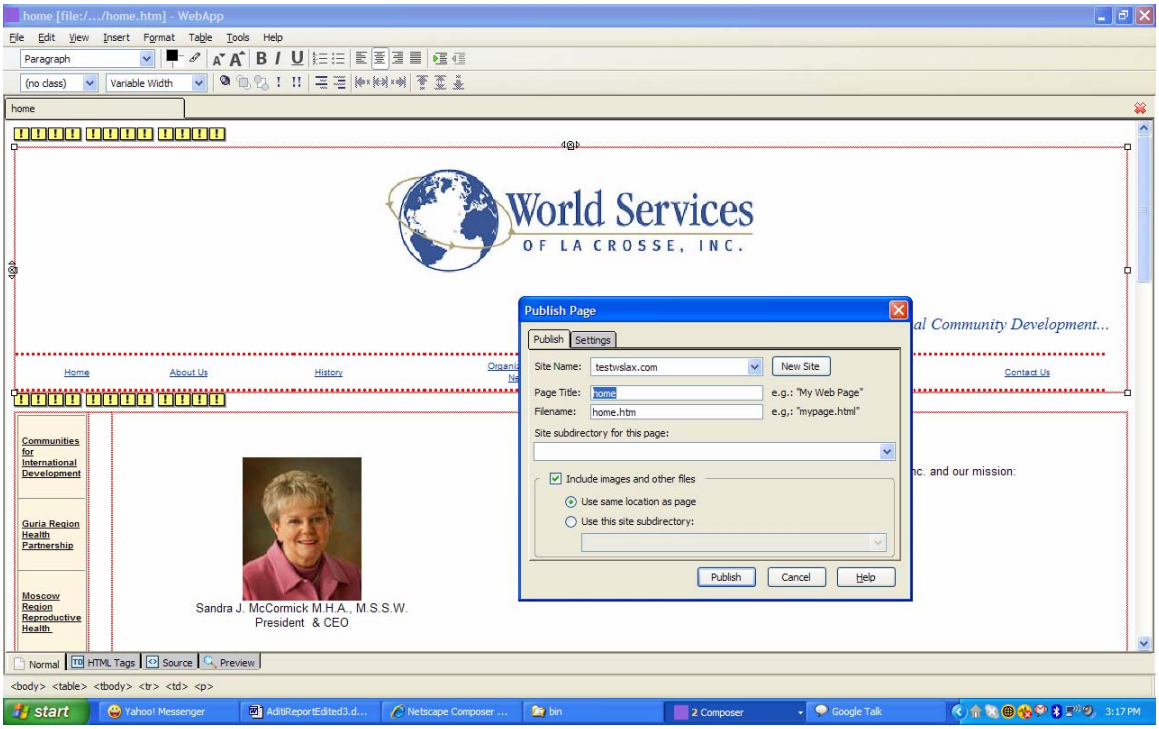


Figure 4.8.2 Normal view of WAT at the time of publishing

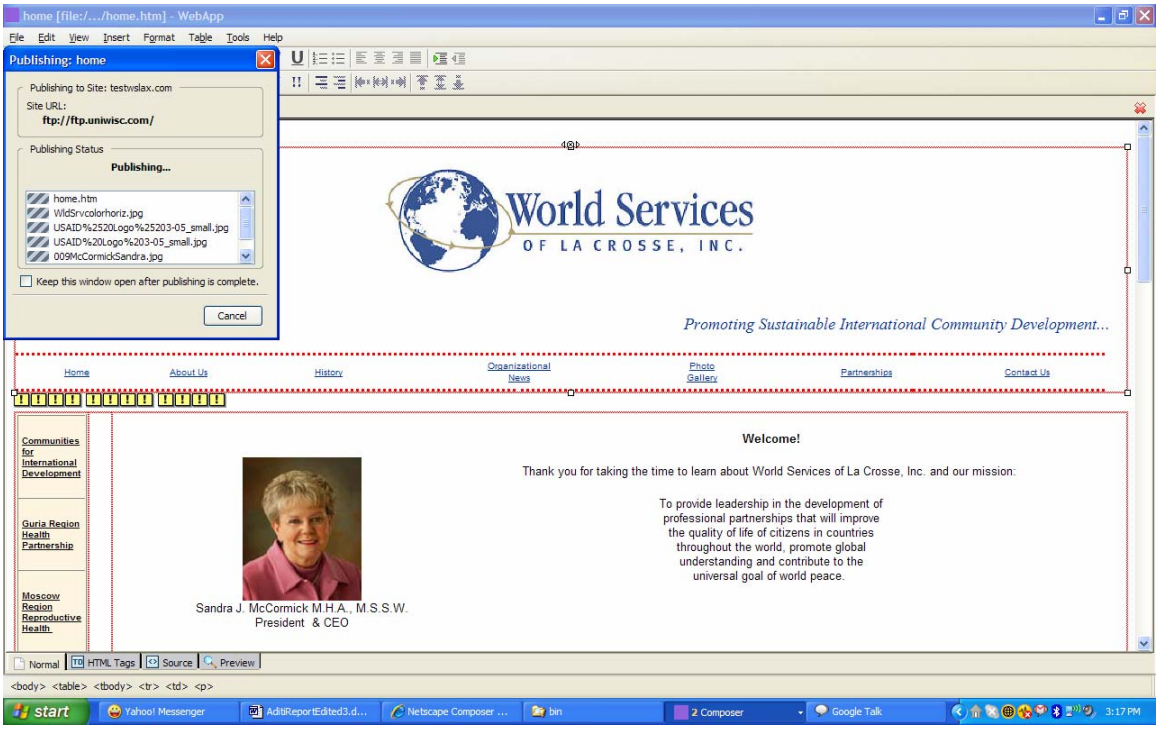


Figure 4.8.3 Normal view of WAT publish is in process

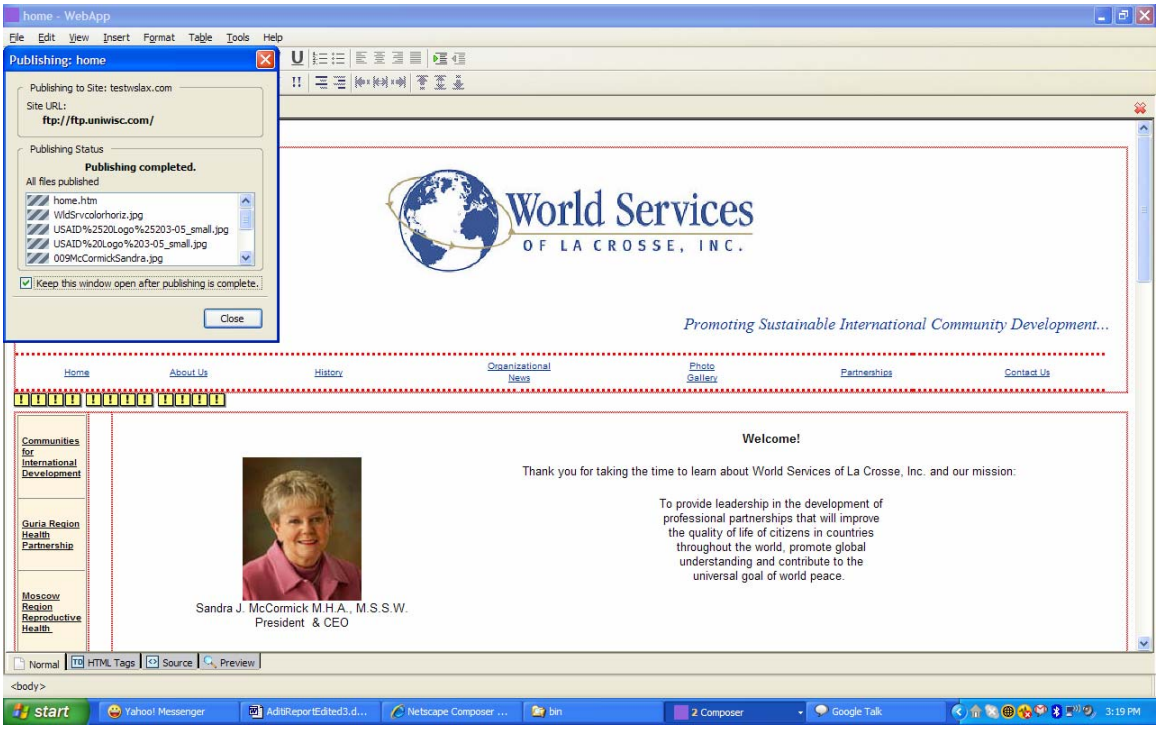


Figure 4.8.4 Normal view of WAT after publishing completed

4.9 Structure used for File management

A common naming scheme is used throughout the Web Application Tool source trees. At the top level is the name of the product (e.g. WebApp). At the second level is the name of the module (e.g. the ‘editor’ directory in WebApp). The third level is where the common naming scheme usually takes over. Many third level directories contain the ‘base’, ‘public’ and ‘idl’ directories. If files can be grouped into submodules, they are usually put in their own third level directory and given a unique name (e.g. ‘txmgr’ in the editor directory in WebApp). Beneath this third level directory, there may be fourth level base, public and idl directories. Thus, the scheme is recursive; it can apply to submodules, sub-submodules and so on.

- *base* contains the basic (core) functionality for the module. Base contains all the source code that cannot be categorized into a submodule.
- *build* contains special makefiles for building the particular modules.
- *doc* contains any documentation associated with this module.
- *idl* contains the XPIDL (Cross Platform Interface Definition Language) interface files. Interfaces are used so functionality can be available to both Javascript scripts and C++ code with as little effort as possible. XPIDL files have their own mini-language and processing tools.
- *public* contains source code that will be exported to the dist/include directory. The source code is not all necessarily public in the general sense; it is sometimes meant for internal use in a certain module. As more code is written with or converted to XPIDL interfaces, the value of the public directory diminishes.
- *src* contains the bulk of the source code.
- *tests* contains C++ harnesses, HTML or XUL that exercise this module.
- *tools* contains scripts for automatically generating certain source code and other special tools for building this module.

5. Limitations

The following are some of the limitations in Web Application Tool:

- **Some functionalities not included**

  Web Application Tool is made for a non-profit organization, as per their requirements. It does not include much in the way of complicated or complex functionalities.

- **No templates included**

  Templates are not supported by Web Application Tool, So when a user wants to create any new website, ready-made templates are not available in the tool. However, the user can save his/her website files as templates.

- **Online help not available**

  The sponsor preferred a written manual to online help. The requested manual was prepared.

- **No drag and drop facility**

  The Web Application Tool does not support a drag and drop facility. This would be a desirable future enhancement.

6. Continuing Work

The following is the continuing work on this project:

- Adding drag and drop facility to make WAT more easy to use for the users. This facility would increase WAT functionality for those users who regularly use this feature in MS Word.
- Currently WAT uses a simple username and password authentication for uploading to the web site, if user wants to save his/her username & password. Adding more security to WAT would make application more secured.
- Web Application Tool could be extended to include some advance functionalities, like spread sheets, templates, and/or the ability to save personalize settings.
- Web Application Tool still need to add the some functionality for the support of different types of web pages.
- An online help facility should be added.

7. Conclusion

This thesis describes the design and implementation of a tool called Web Application Tool. WAT assists users with less or no knowledge of any computer programming language to create a new website or edit an existing website.

Core Functionalities:

- Add, update and delete text
- Add, resize and delete images
- Add, update and delete hyperlinks
- Add, update and delete tables
- Table management
- Form management
- Support of CSS (Cascading Style Sheets) to help users to define colors, fonts, layout, and other aspects of document presentation.
- Standalone WYSIWYG editor
- Built-in publishing with FTP
- View and edit HTML source
- Support for opening multiple websites

8. Bibliography

[1] I. Sommerville, *Software Engineering 6th Ed.*, Addison Wesley, 2001

[2] A.M. Davis, *Software Requirements Analysis and Specification,* Prentice Hall, 1990.

[3] (March 2005) "Chapter 2: Software Requirements", in Executive editors: Alain Abran, James W. Moore; editors Pierre Bourque, Robert Dupuis: *Guide to the software engineering body of knowledge*, 2004 Version, Los Alamitos, CA: IEEE Computer Society Press. ISBN 0-7695-2330-7. Retrieved on 2007-02-08. "It is widely acknowledged within the software industry that software engineering projects are critically vulnerable when these activities are performed poorly."

[4] Computer Society/Software and Systems Engineering Standards Committee, "Recommended Practice for Software Requirements Specifications", Std. 830-1998, IEEE Standards Association, Piscataway, NJ, September 16, 1997

[5] J. Buyens, Microsoft FrontPage Version 2002 Inside Out, Microsoft Press, 2001.

[6] I. Sommerville, *Software Engineering 6th Ed.*, Addison Wesley, 2001

[7] Hans van Vliet, Software Engineering: Principles and Practice, John Wiley & Sons, Ltd., 2000

[8] T. Powell, HTML & XHTML: The Complete Reference 4th Ed., McGraw-Hill Osborne Media, 2003.

[9] www.waterproof.fr, PHPEdit, Waterproof Software, 2003.

[10] P. Whitehead and J. Desamero, PHP: Your Visual Blueprint for Creating Open Source, Server-Side Content, Hungry Minds Inc., 2001.

APPENDIX A: How HTML parsing works using the Gecko Engine

HTML

This document describes the complete handling of HTML by the Gecko engine. Gecko manages the parsing process - how HTML is lexically analyzed and then interpreted. After the parsing process is discussed we give a detailed analysis of each HTML tag and the attributes that are supported.

Parsing

HTML is tokenized by an HTML scanner. Stream converters are used to translate from various encodings to Unicode. The scanner separates the Unicode stream into tokens which consist of:

- text
- HTML tags
- entities
- script-entities
- comments
- conditional comments

The parsing engine analyzes tokens produced by the scanner in a set of well defined steps:

- The parser processes the head portion of the document first, without emitting any output. This is done to discover a few special features of html:
    - The parser processes META tags looking for META TARGET
    - The parser processes META tags looking for META tags which affect the character set. Nav4 is a standalone indexer/search engine or integrated with existing search engine as "patch kit", which handles the very first character set defining meta tag (all others are ignored) by reloading the

document with the proper character conversion module inserted into the stream pipeline.

- After the head portion is processed, the parser then proceeds to process the body of the document

HTML Tag Processing

Tags are processed by the parser by locating a **tag handler** for the tag. The HTML parser serves as the tag handler for all of the built-in tags documented below. Tag attribute handling is done during translation of tags into content. This mapping translates the tag attributes into content data and into style data. The translation to style data is documented below by indicating the mapping from tag attributes to their CSS1 (plus extensions) equivalents.

The following list describes hacks added to the parsing engine to deal with navigator compatibility. These are just the parser hacks, not the layout or presentation hacks. Most hacks are introduced for HTML syntax error recovering. HTML doesn't specify much how to handle those error conditions. Mozilla has made big effort to render pages with non-prefect HTML. For many reasons, new browsers need to keep compatible in this area.

- Entities can be used as escape in quoted string. For value string in name-value pair. Test line 70 shows that an entity quote at the beginning means the value is NOT quoted. Test line 90 shows that if the value is started with a quote, then an entity quote does NOT terminate the value string.
- Wrapping tags are special tags such as title, text area, server, script, style, and etc.. The comment in ns\lib\libparse\pa_parse.c says:

  /*

  * These tags are special in that, after opening one of them, all other tags are ignored until the matching

  * closing tag.

  */

During the searching of an end tag, comments and quoted strings are observed. 6.0 handles comments now, need to add quoted string.

- If a <tr> or <td> tag is seen outside any <table> scope, it is ignored.
- In case of table in table but not in cell, table tags before the last table tag are ignored. For example, <table> <table border> .....,or <table> <tr> <table border>..., the table will be displayed with border. There table and tr tags are buffered for this recovery. When a TD or CAPTION tag is open, the buffer is flushed out, because we cannot buffer contents of TD or CAPTION for performance and memory constrains. They are sub documents and can be very big. If we see a <table> outside cell after previous table is flushed out, the new <table> tag is ignored. Nav4.0 can discard previous table in such case.
- Caption is not a commonly used feature. In this, captions can be anywhere. For Captions outside cells, the first one takes effect. For captions inside cells, the last one takes effect, and they also close TD and TR. In 6.0, caption is limited to the standard position: after <table>. Captions in other places are ignored; their contents are treated as text. See test case table05a.html to table05o.html.
- For <table> <tr> <tr>, the first <tr> takes effect.
- The nav4 parser notices when it hits EOF and it's in the middle of scanning in a comment. When this happens, the parser goes back and looks for an improperly closed comment (e.g. a simple > instead of a -->). If it finds one, it represses the input after closing out the comment.
- When Nav4.0 sees the '<' sign, it searches for '>', observing quoted values. If it cannot find one till EOF, the '<' sign is treated as text. In Xena 6.0, a limit is set for how far the '>' is searched. The default limit is 4096 char, and there is a API HTMLScanner.setMaxTagLength() to changed it setting -1 means no limit, which is same as Nav4.0.

Head objects:

TITLE

The TITLE tag is a container tag whose contents are not HTML. The contents are pure text and are processed by the parser until the closing tag is found. There are

no attributes on the tag and any white space present in the tag is compressed down with leading and trailing white space eliminated. The first TITLE tag found by the parser is used as the document's title (subsequent tags are ignored).

BASE

This sets the base element in the head portion of the document and it also defines the base URL for all links in the document.

Attributes:

HREF=url [This is an absolute URL]

TARGET=string [must start with XP_ALPHA|XP_DIGIT|underscore otherwise nav4 ignores it]

META

This defines several header fields (content-encoding, author, etc.)

Attributes:

REL=SMALL_BOOKMARK_ICON|LARGE_BOOKMARK_ICON

SRC=string

HTTP-EQUIV="header: value"

CONTENT=string

HTTP-EQUIV values (from libnet/mkutils.c NET_ParseMimeHeader):

ACCEPT-RANGES

CONTENT-DISPOSITION

CONTENT-ENCODING

CONTENT-RANGE

CONTENT-TYPE [defines character set only]

CONNECTION

DATE

EXPIRES

EXT-CACHE

LOCATION

LAST-MODIFIED

LINK

PROXY-AUTHENTICATE

PROXY-CONNECTION

PRAGMA

RANGE

REFRESH

SET-COOKIE

SERVER

WWW-AUTHENTICATE

WWW-PROTECTION-TEMPLATE

WINDOW-TARGET

Style sheets and HTML w3c spec adds this:

CONTENT-STYLE-TYPE

LINK

Links are used by extensions mechanism to find tag handlers.

Attributes:

REL=FONTDEF

SRC=url

REL=STYLESHEET [If MEDIA param is defined it must ==nc screen]

LANGUAGE=LiveScript|Mocha|JavaScript1.1|JavaScript1.2

TYPE="text/javascript" | "text/css"

HREF=url

ARCHIVE=url

CODEBASE=url

ID=string

SRC=url

Note: HREF takes precedence over SRC in Nav4.

HEAD

/HEAD clears the "in_head" flag (but leaves the "in_body" flag alone.

Text in head clears in_head, and set in_body true, just as if the author forgot the /HEAD tag.

Attributes: none

HTML

Attributes: none

STYLE

Styles are allowed anywhere in the document. Only the entities are not parsed in the style tag's content.

Attributes:

LANGUAGE=LiveScript|Mocha|JavaScript1.1|JavaScript1.2

TYPE="text/javascript" | "text/css"

HREF=url

ARCHIVE=url

CODEBASE=url

ID=string

SRC=url

FRAMESET

Frameset is a set of rows & columns available in a frame. For frameset a default property has been set as rows=1 and cols=1.

Attributes:

FRAMEBORDER= no | 0 (zero) [default is no_edges=false]

BORDER= int [clamped: >= 0 && <= 100]

BORDERCOLOR= color

ROWS= pct-list

COLS= pct-list

FRAME

Frame contains a border and the default value of border has been set as width of zero.

Attributes:

FRAMEBORDER= no | 0 (zero) [default is framesets value]

BORDER= int [clamped; >= 0 && <= 100]

BORDERCOLOR= color

NORESIZE= true [default is false]

SCROLLING= yes | scroll | on | no | noscroll | off

SRC= url [clamped: prevent recursion by eliminating any ancestor

references]

NAME= string

MARGINWIDTH= int (clamped: >= 1)

MARGINHEIGHT= int (clamped: >= 1)

NOFRAMES

Nonframes are generally used when frames are disabled or for backrev browsers. It has no stylistic consequences.

Body objects:

BODY

The tag is only processed on open tags and it is always processed. See ns\lib\layout\laytags.c, searching for "case P_BODY". During tag processing, the in_head flag is set to false and the in_body flag is set to true. An attribute is ignored if the document already has that attribute set. Attributes can be set by style sheets, or by previous BODY tags.

Attributes:

MARGINWIDTH=int [clamped: >= 0 && < (windowWidth/2 - 1)]

MARGINHEIGHT=int [clamped: >= 0 && < (windowHeight/2 - 1)]

BACKGROUND=url

BGCOLOR=colorspec

TEXT=colorspec

LINK=colorspec

VLINK=colorspec

ALINK=colorspec

ONLOAD, ONUNLOAD, UNFOCUS, ONBLUR, ONHELP=script

ID=string

LAYER, ILAYER

Open layer/ilayer tag automatically close out an open form if any form is open.

Attributes:

LEFT=value-or-pct (pct of `right-left` margin)

PAGEX=x (if no LEFT)

TOP=value-or-pct

PAGEY=y (if no TOP)

CLIP=clip

WIDTH=value-or-pct (pct of `right-left` margin)

HEIGHT=value-or-pct

OVERFLOW=string

NAME=string

ID=string

ABOVE=string

BELOW=string

ZINDEX=int [any value]

VISIBILITY=string

BGCOLOR=colorspec

BACKGROUND=url

NOLAYER

Nolayer container is used when layers are disabled or unsupported. The nolayer container has no style consequences by default.

P

P is used to close the paragraph. If the attribute is present, then an alignment gets pushed on the alignment stack.

Attributes:

ALIGN=divalign

ADDRESS

ADDRESS closes out the open paragraph. The open tag does a conditional soft line break and then pushes a merge of the current style with italics enabled onto the style stack. The close always pops the style stack and also does a conditional soft line break.

PLAINTEXT, XMP

PLAINTEXT causes the remaining content to no longer be parsed. XMP causes the content to not parse entities or other tags. The XMP can be closed by it's own tag (on any boundary); PLAINTEXT is not closed (html3.2 allows it to be closed). Both tags change the style to a fixed font of a

LISTING

Listing does a hard line break on open and close. Open pushes a fixed width font style of a particular font size on the style stack. The close tag pops the top of the style stack.

Attributes: none

PRE

Pre first closes the paragraph. If a tag is open it does a hard line break followed by fixed font style (unless VARIABLE is present) is pushed on the style stack. The close tag pops the top of the style stack with a hard line break.

Attributes:

WRAP

COLS=int [clamped: >= 0]

TABSTOP=int [clamped: >= 0; clamped value is replaced with default value]

VARIABLE

NOBR

NOBR sets or clears a flag in the state machine. It has no effect on any other state. This tag doesn't nest.

CENTER

Center is used to align the text. It always does a conditional soft line break. The open tag pushes an alignment on the alignment stack. The close tag pops the top alignment off.

Attributes: none

DIV

It always does a conditional soft line break. COLS define the number of columns to layout in (like MULTICOL). The open tag pushes an alignment on the alignment stack (if COLS > 1 then it pretends to be a MULTICOL tag). The close

tag pops an alignment from the alignment stack.

Attributes:

ALIGN=divalign

COLS=int [if cols > 1 then DIV acts like a MULTICOL tag else DIV is just a container]

GUTTER= int (clamped: >= 1)

WIDTH= value-or-pct [pct of right-left margin; clamped >= 1/0 (strange code)]

H1-H6

The open tag does a hard line break and pushes a style item which enables bold and disables fixed and italic. The close tag always pops the top item from the style stack. It also does a hard line break. If the **ALIGN** attribute is present then the open tag pushes an alignment on the alignment stack. The close tag will look at the top of the alignment stack and if its a header of any kind (H1 through H6) then the alignment is popped. In either case the close tag also does a conditional soft line break (this happens before the hard line break).

Attributes:

ALIGN=divalign

A note regarding closing paragraphs: Any time a close paragraph is done (for any tag) if the top of the alignment stack has a tag named "P" then a conditional soft line break is done and the alignment is popped.

TABLE

Attributes:

ALIGN=left|right|center|abscenter

BORDER=int [clamped: if null then -1, if < 1 then 1]

BORDERCOLOR=string [if not supplied then set to the text color]

VSPACE=int [clamped: >= 0]

HSPACE=int [clamped: >= 0]

BGCOLOR=color

BACKGROUND=url

WIDTH=value-or-pct [% of win.width minus margins; clamped: >= 0]

HEIGHT=value-or-pct [% of win.height minus margins; clamped: >= 0]

CELLPADDING=int [clamped: >= 0; separate pads take precedence]

TOPPADDING= int [clamped: >= 0]

BOTTOMPADDING= int [clamped: >= 0]

LEFTPADDING= int [clamped: >= 0]

RIGHTPADDING= int [clamped: >= 0]

CELLSPACING= int [clamped: >= 0]

COLS=int [clamped: >= 0]

The code supports more attributes in the Table attribute handler than it does in the code that gets the attributes from the tag! They are border_top, border_left, border_right, border_bottom, border_style (defaults to outset; allows for outset/dotted/none/dashed/solid/double/groove/ridge/inset).

TR

TR means Table Row and if it is open then it automatically closes an open table row (and an open table cell if one is open). It also automatically closes a CAPTION tag.

Attributes:

BGCOLOR=color

BACKGROUND=url

VALIGN=top|bottom|middle|center(==middle)|baseline; default is top

ALIGN=left|right|middle|center(==middle); default is left

TH, TD

If no table, then the tag is ignored (open or close). If no row is currently opened or the current row is current done (because of a </TR> tag), then a new row is begun. The tag parameters for the row come from the TH/TD tag in if no row is opened. An open of either of these tags will automatically close the previous cell.

Attributes:

COLSPAN=int [clamped: >= 1 && <= 1000]

ROWSPAN=int [clamped: >= 1 && <= 10000]

NOWRAP [boolean: disables wrapping]

BGCOLOR=color [default: inherit from the row; if not row then table; if not table then inherit from an outer table cell; this works because the style is flattened so the outer table cell will have a color]

BACKGROUND=url [same rules as bgcolor for inheritance; tile mode is inherited too and not settable by TH/TD attributes (have to use style sheets for that)]

VALIGN=top|bottom|middle|center(==middle)|baseline; default is top

ALIGN=left|right|middle|center(==middle); default is left

WIDTH=value-or-pct [clamped: >= 0]

HEIGHT=value-or-pct [clamped: >= 0]

CAPTION

An open caption tag will automatically closes an open table row (and an open cell).

Attributes:

ALIGN=bottom

The code sets the vertical alignment to top w/o providing a mechanism for the user to set it (there is no VALIGN attribute).

MULTICOL

The open tag does a hard line break. The close tag checks to see if the state machine has an open multiple columns and if it does then it does a conditional soft line break and then continues to break until both margins are cleared of floating elements. It re compute the margins based on the list indenting level. After the synthetic table is output the close tag does a hard line break.

This tag will treat the input as source for a table with one row and COLS columns. The data is laid out using the width divided by the number of columns. After the total height is known, the content is partitioned as evenly as possible between the columns in the table.

Attributes:

COLS=int [clamped: values less than 2 cause the tag to be ignored]

GUTTER=int [clamped: >= 1]

WIDTH=value-or-pct [pct of right-left margin; clamped: >= 1/0 (strange code)]

BLOCKQUOTE

The open tag does a hard line break. A list with the empty-bullet style is pushed on the list stack (unless TYPE=cite/jwz then a styled list is pushed). The close tag pops any list and does a hard line break.

Attributes:

TYPE=cite | jwz

UL, OL, MENU, DIR

For top-level lists (lists not in lists) a hard break is done on the open tag, otherwise a conditional-soft-break is done. Tag always does a close paragraph. The close tag does a conditional soft line break when nested; when not nested the close tag does a hard line break (even if no list is open). The open tag pushes the list on the list stack. The close tag pops any list off the list stack. Attributes:

TYPE= none | disc | circle | round | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | A | a | I | i [clamped: if none of the above is picked and OL then the bullet type is "number" otherwise the bullet type is "basic"]

START=int [clamped: >= 1]

COMPACT

DL

For the open tag, if the list is nested then a conditional soft line break is done otherwise a hard line break is done. The open tag pushes a list on the list stack. The close tag pops any list from the list stack. Closing the list acts like other lists closes.

Attributes:

COMPACT

LI

The open tag does a conditional soft line break. Close tags are ignored (except for closing the paragraph).

Attributes:

TYPE= A | a | I | i (if the containing list is an **OL**)

TYPE= round | circle | square (if the containing list is not **OL** and not **DL**)

VALUE=int [clamped: >= 1]

The html parser allows the full set of list item styles from the OL/DL tag instead of just the limited set that nav4 allows.

DD

Close tags are ignored (except for closing the paragraph). DD outside a DL just advances the X coordinate of layout by a small constant. DD inside a DL does a conditional soft line break and other margin crud.

Attributes: none.

DT

Close tags are otherwise ignored. It does a conditional soft line break. Moves the X layout coordinate to the left margin.

Attributes: none

A

Open anchors push a style on the style stack, if the anchor has an **HREF**. Close anchors pop as many styles off the top of the style stack that are anchor tags (anchor tags don't nest in other words). In addition, any styles on the stack that has the ANCHOR bit set have it cleared and fiddle with the foreground and background colors.

Attributes:

NAME=string

HREF=url

TARGET=target

SUPPRESS=true

STRIKE, S, TT, CODE, SAMPLE, KBD, B, STRONG, I, EM, VAR, CITE, BLINK, BIG, SMALL, U, INLINEINPUT, SPELL

The open tag pushes onto the style stack. The close tag always pops the top item from the style stack.

Attributes: none

SUP, SUB

The open tag pushes a font size decrease on the style stack. The close tag always pops the top of the style stack. The open and close tag impacts the baseline. The only difference between SUP and SUB is how they impact the baseline. Note that the baseline information is forgotten after a line break; therefore a close SUP/SUB on the next line will do strange things.

Attributes: none

SPAN

Ignored by the navigator.

Attributes: none

FONT

The open font tag with no attributes resets the font size to the base font size. The open tag always pushes a style stack entry. The close tag always pops the top item off the style stack.

Attributes:

SIZE=[+ int | - int | int] [clamped: >=1 && <= 7]

POINT-SIZE=[+ int | - int | int] [clamped: >= 1 && <= 1600]

FONT-WEIGHT=[+ int | - int | int] [clamped: >= 100 && <= 900]

COLOR=colorspec

FACE=string

A note regarding the style stack: The pop of the stack checks to see if the top of the stack is an ANCHOR tag. If it is not an anchor then the top item is unconditionally popped. If the top of the style stack is an anchor tag then the code searches for either the bottom of the stack or the first style stack entry not created by an anchor tag. If the entry is followed

by another entry then the entry is removed from the stack (an out-of-order pop in other words). In this case the anchor style stack entry is left untouched.

text, entities

These are basic content objects that get fed directly to the output. In navigator the text is processed by doing line-breaking (entities have been converted to text already by the parser). The line-breaking is controlled by the margin settings and the list depth, the floating elements, the style attributes (font size, etc.), the preformatted flag, the no-break flag and so on.

IMG, IMAGE

Close tag is ignored.

Attributes:

ISMAP

USEMAP=url

ALIGN=alignparam

SRC=url [whitespace is stripped]

LOWSRC=url

ALT=string

WIDTH=value-or-pct (pct of `right-left` width)

HEIGHT=value-or-pct (pct of window height)

BORDER=int [clamped: >= 0]

VSPACE=int [clamped: >= 0]

HSPACE=int [clamped: >= 0]

SUPPRESS=true | false

HR

If an open tag then does a conditional soft line break. The rule inherits alignment from the parent container unless there is no container (then it's centered) or if the tag defines it's own alignment. After the object is inserted into the layout stream a soft line break is inserted as well.

Attributes:

ALIGN=divalign (sort of; in laytags.c it's divalign; in layhrule.c it's left or right only)

SIZE=int (1 to 100 inclusive)

WIDTH=val-or-pct (pct of `right-left` width)

NOSHADE

BR

BR is used for Break and it does an unconditional soft break. If clear is set then it will also soft break until either the left or right or both margins are clear of floating elements. Note that /BR == BR!

Attributes:

CLEAR=left | right | all | both

WBR

WBR means a soft word break.

Attributes: none

EMBED

Close tag does nothing. Embed's operate inline just like images (they don't close the paragraph).

Attributes:

HIDDEN=no | false | off

ALIGN=alignparam

SRC=url

WIDTH=val-or-pct (pct of `right-left` width)

HEIGHT=val-of-pct; if val is < 1 (sometimes) the element gets HIDDEN automatically

BORDER=int (unsupported by navigator)

VSPACE=int [clamped: >= 0]

HSPACE=int [clamped: >= 0]

NOEBMED

Used when EMBED's are disabled. It is a container for regular content that has no stylistic consequences (no line breaking, no style stack effect, etc.).

APPLET

Applet tags don't nest (there is a notion of current_applet). The open tag automatically closes an open applet tag.

Attributes:

ALIGN=alignparam

CODE=string

CODEBASE=string

ARCHIVE=string

MAYSCRIPT

NAME=string [clamped: white space is stripped out]

WIDTH=value-or-pct [pct of right-left width; clamped: >= 1]

HEIGHT=value-or-pct [pct of window height; clamped >= 1]

BORDER=int [clamped: >= 0]

HSPACE=int [clamped: >= 0]

VSPACE=int [clamped: >= 0]

If no width is provided:

if a height was provided, use the height. Otherwise, use 90% of the window width if percentage widths are allowed; otherwise use a value of 600.

If no height is provided:

if a width was provided, use the width. Otherwise, use 50% of the window height if percentage widths are allowed; otherwise use a value of 400.

If the applet is hidden, then the widht/height gets forced to zero.

PARAM

The param tag is supported when contained by the APPLET tag or the OBJECT tag. It has no stylistic consequences. The attribute values from the tag are passed to the containing APPLET or OBJECT. Note that /PARAM == PARAM.

Attributes:

NAME=string [clamped: white space is stripped out]

VALUE=string [clamped: white space is stripped out]

White space being stripped is done as follows: leading and trailing whitespace is removed. Any embedded whitespace is left alone except if it's a non-space whitespace in which case it is removed.

OBJECT

The open tag pushes an object onto the object stack. The close tag pops from the object stack. I don't understand how the data stuff works.

Attributes:

CLASSID=string (clsid:, java:, javaprogram:, javabean: are the supported prefixes; maybe it's a url if no prefix shown)

TYPE=string (a mime type)

DATA=string (data: prefix mentions a url)

There are more attributes that depend on the type of object being embedded in the page. If the object is a java bean then the applet parameters are supported:

CLASSID

HIDDEN

ALIGN

CLASSID (instead of CODE)

CODEBASE

ARCHIVE

MAYSCRIPT

ID (applets use NAME)

WIDTH

HEIGHT

BORDER

HSPACE

VSPACE

MAP

The open tag automatically closes an open map (maps don't nest). There is no stylistic consequence of the map nor does it provide any visible presentation in the normal layout case (an editor would do something different). The map can be

declared anywhere in the document.

Attributes:

NAME=string [clamped: white space is stripped out]

AREA

Does nothing if there is no current map or the tag is a close tag.

Attributes:

SHAPE=default | rect | circle | poly | polygon

ALT=string [clamped: newlines are stripped]

COORDS=coord-list

HREF=url

TARGET=target (only if HREF is specified)

SUPPRESS

SERVER

A container for server-side javascript. Not evaluated by the client (parsed and ignored). Note: The navigator parser doesn't expand entities in a **SERVER** tag.

SPACER

Open spacer tag provides whitespace during layout: **TYPE**=line/vert/vertical causes a conditional soft line break and then adds **SIZE** to the Y layout coordinate. **TYPE**=word causes a conditional soft word break and then adds **SIZE** to the X layout coordinate. **TYPE**=block causes blockish layout stuff to happen.

Attributes:

TYPE=line | vert | vertical | block (default: word)

ALIGN=alignparam (these 3 params are only for **TYPE**=block)

WIDTH=value-or-pct

HEIGHT=value-or-pct

SIZE=int [clampled: >= 0]

SCRIPT

Note: The navigator parser doesn't expand entities in a SCRIPT tag.

Attributes:

LANGUAGE=LiveScript | Mocha | JavaScript1.1 | JavaScript1.2

TYPE="text/javascript" | "text/css"

HREF=url

ARCHIVE=url

CODEBASE=url

ID=string

SRC=url

NOSCRIPT

Used when scripting is off or by backrev browsers. It is a container that has no stylistic consequences.

FORM

Attributes:

ACTION=href

ENCODING=string

TARGET=string

METHOD=get | post

ISINDEX

This tag is a shortcut for creating a form element with a submit button and a single text field. If the PROMPT attribute is not present in the tag then the value used is **"This is a searchable index. Enter search keywords:"**.

Attributes:

PROMPT=string

ACTION=href

ENCODING=string

TARGET=string

METHOD=get | post

INPUT

Attributes vary according to type:

TYPE= text | radio | checkbox | hidden | submit | reset | password | button | image | file | jot | readonly | object

NAME= string

TYPE=image

attributes are from the IMG tag (!)

TYPE= text | password | file

font style is forced to fixed

VALUE= string

SIZE= int (clamped; >= 1)

MAXLENGTH= int (not clamped!)

TYPE= submit | reset | button | hidden | readonly

VALUE=string; default if no value to the attribute varies according to the type:

submit -> "Submit Query"

reset -> "Reset"

others -> " " (2 spaces)

Note also that the value has newlines stripped from it

WIDTH=int (clamped >=0 && <= 1000) (only for submit, reset or button)

HEIGHT=int (clamped >=0 && <= 1000) (only for submit, reset or button)

TYPE=radio | checkbox

CHECKED (flag - if present then set to true)

VALUE= string (the default value is "on")

SELECT

Attributes:

MULTIPLE (boolean)

SIZE= int (clamped >= 1)

NAME= string

WIDTH= int (clampled >= 0 && <= 1000)

HEIGHT= int (clamped >= 0 && <= 1000; only examined for single entry lists (!multiple || size==1))

OPTION

Lives inside the SELECT tag (ignored otherwise).

Attributes:

VALUE=string

SELECTED boolean

TEXTAREA

Attributes:

NAME=string

ROWS=int (clamped; >= 1)

COLS=int (clamped; >= 1)

WRAP= off | hard | soft (default is off; any value which is not known turns into soft)

KEYGEN

Attributes:

NAME=string

CHALLENGE=string

PQG=string

KEYTYPE=string

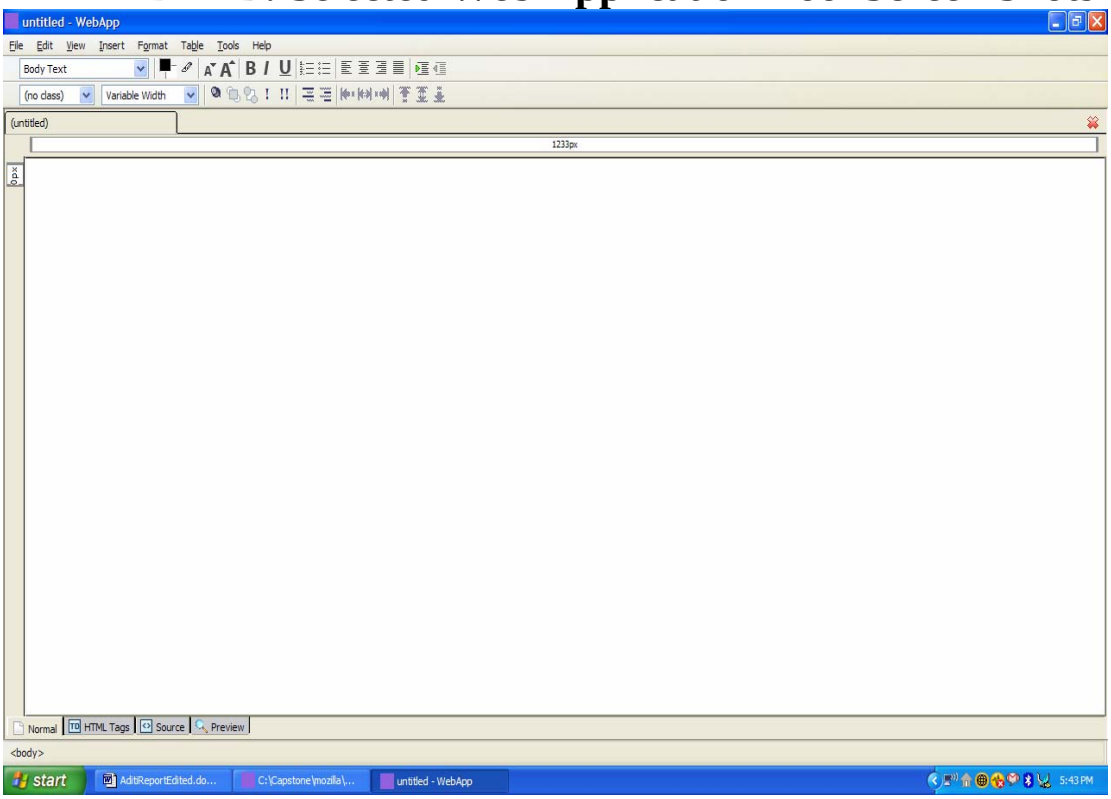
BASEFONT

Sets the base font value which +/- size values in FONT tags are relative to.

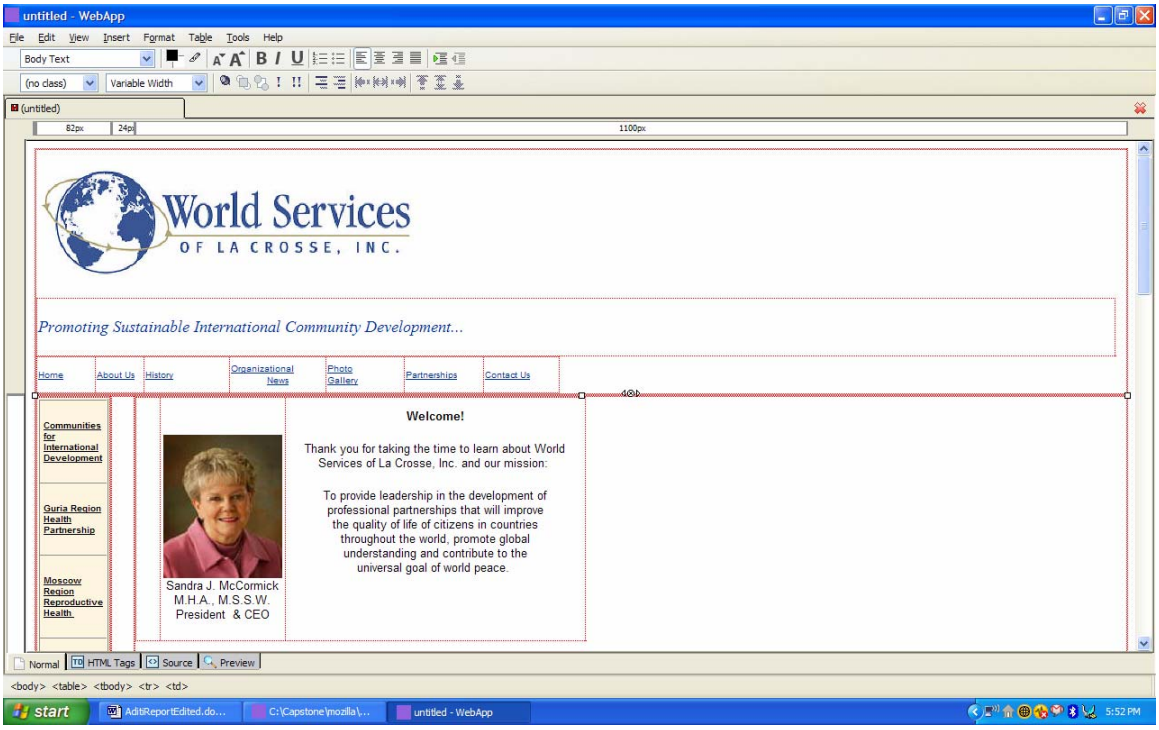
Attributes:

SIZE=+ int | - int | int (just like FONT)

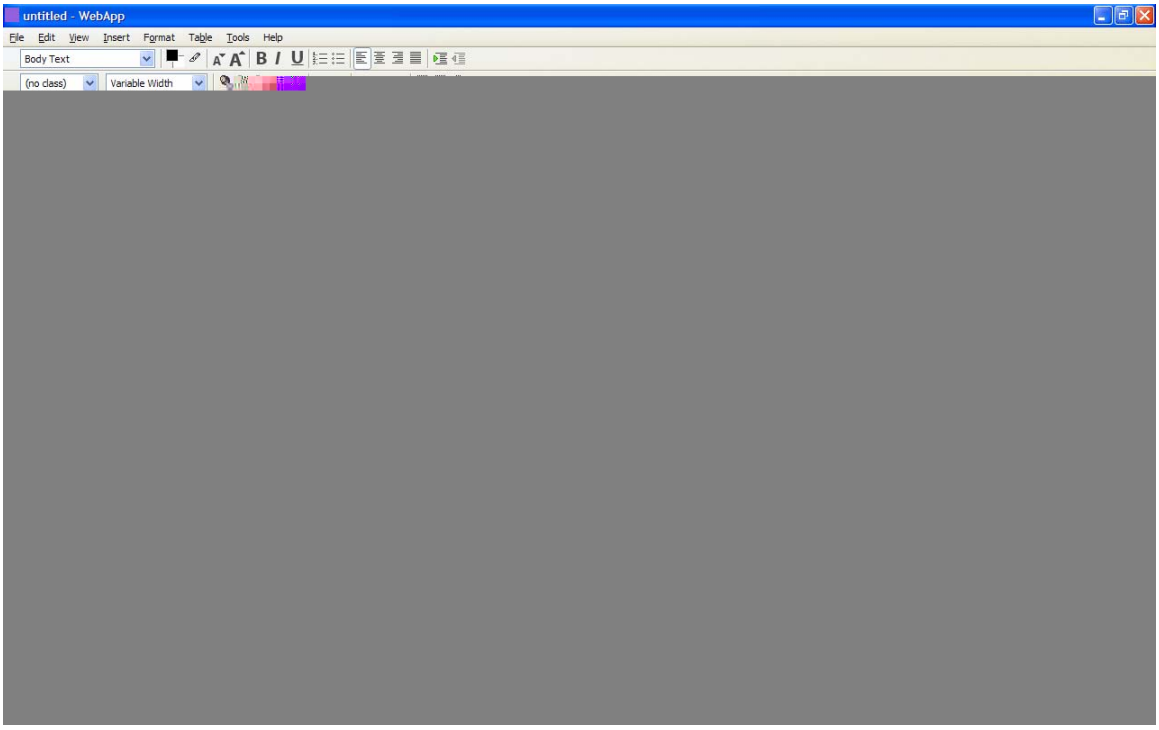
APPENDIX B: Selected Web Application Tool Screen Shots



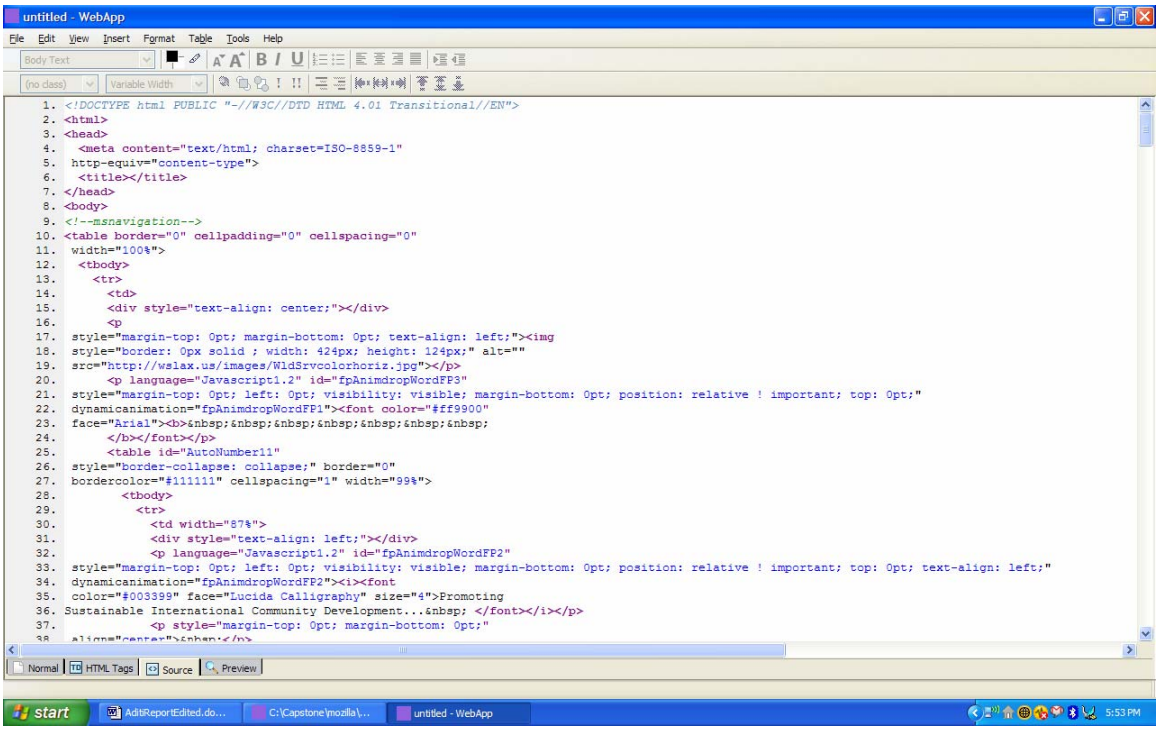
blank page



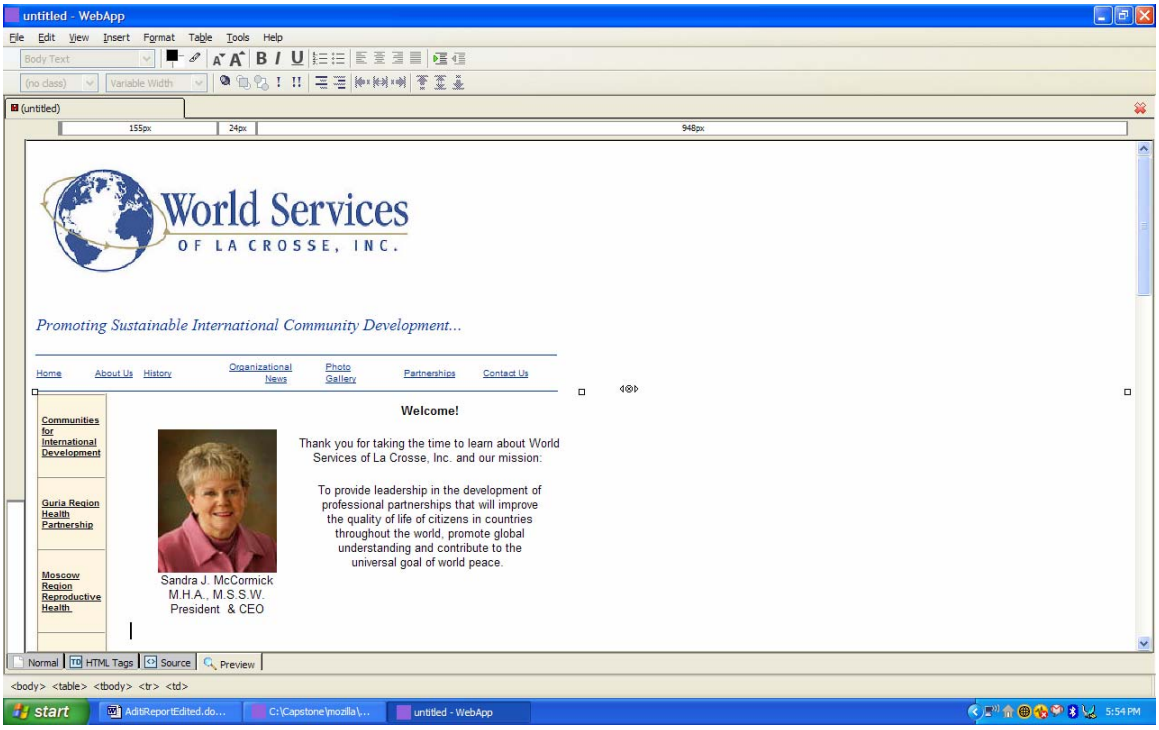
Open an existing Web site in Normal view



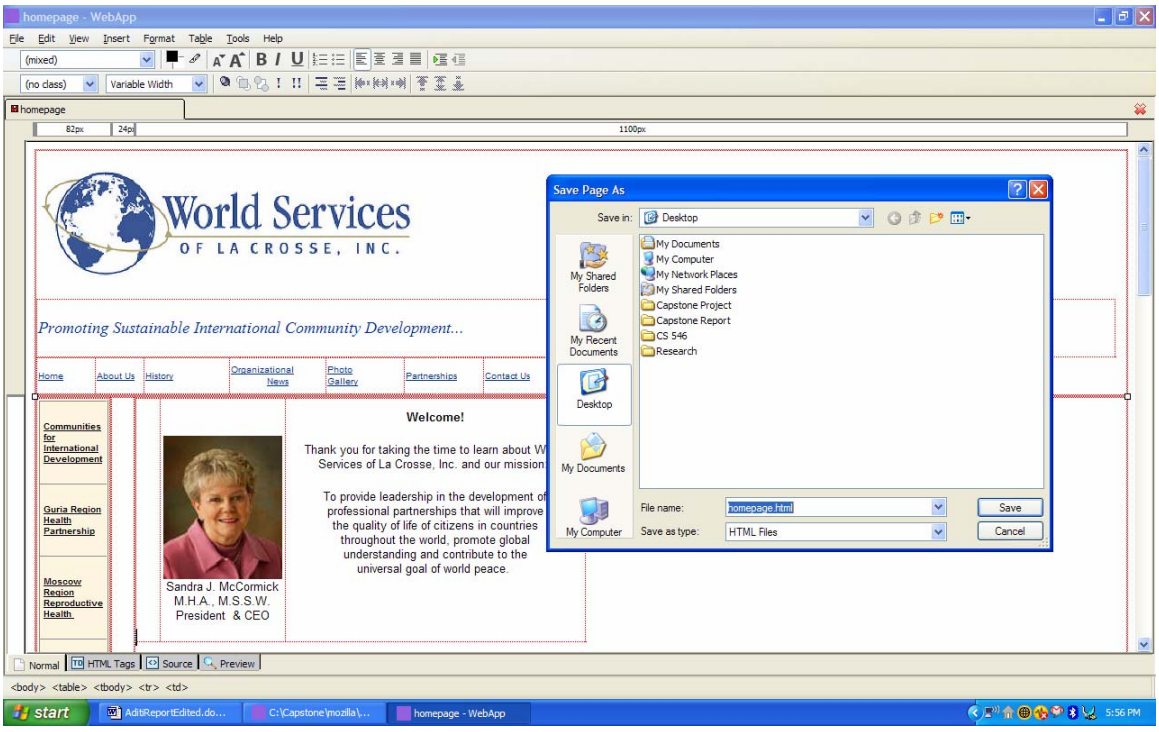
HTML Tag view



Source view

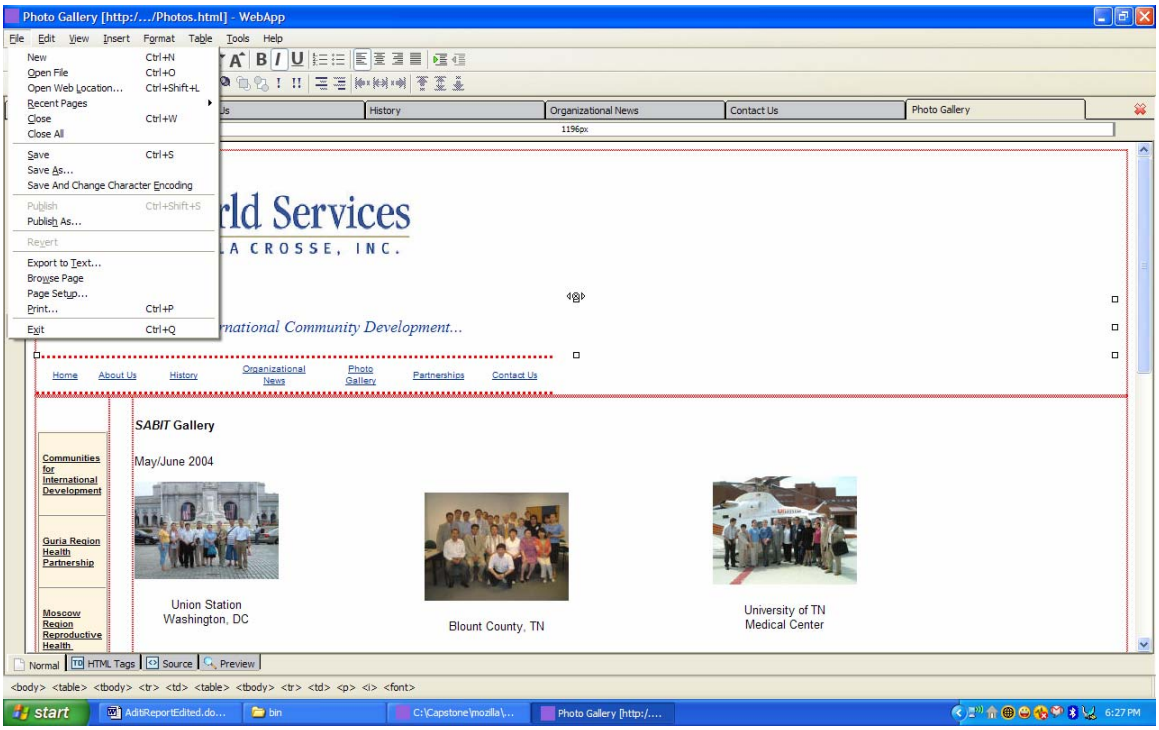


Preview

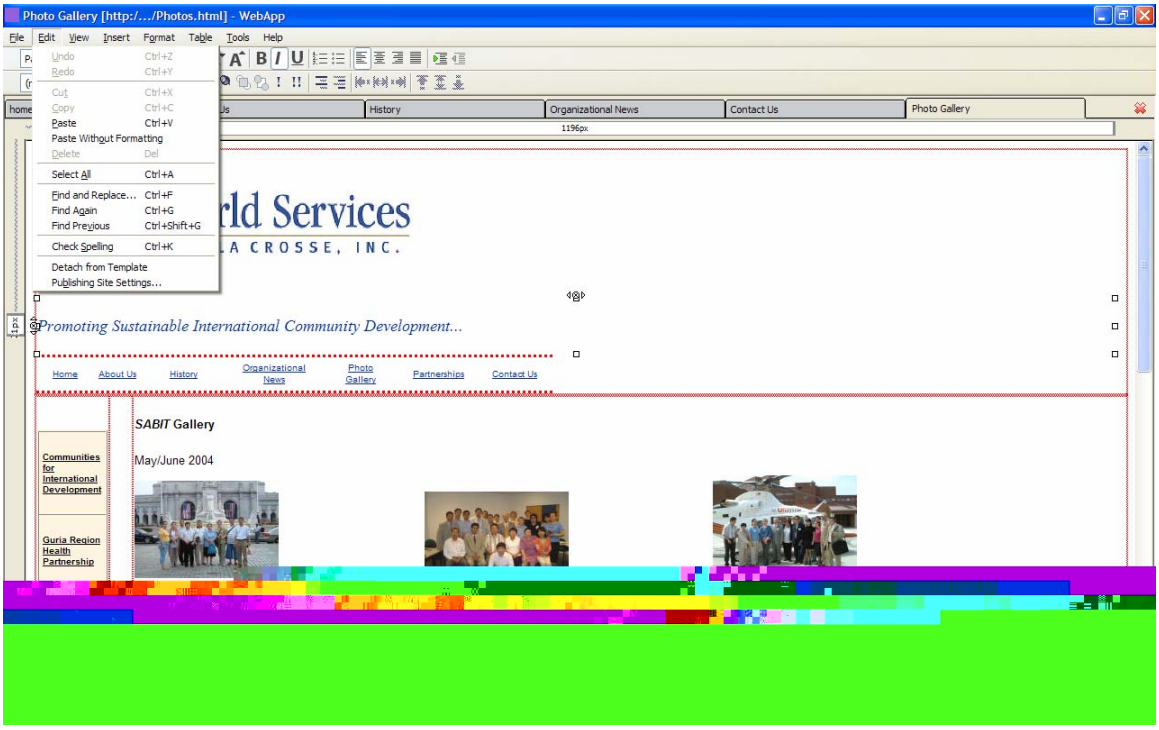


Saving web Page

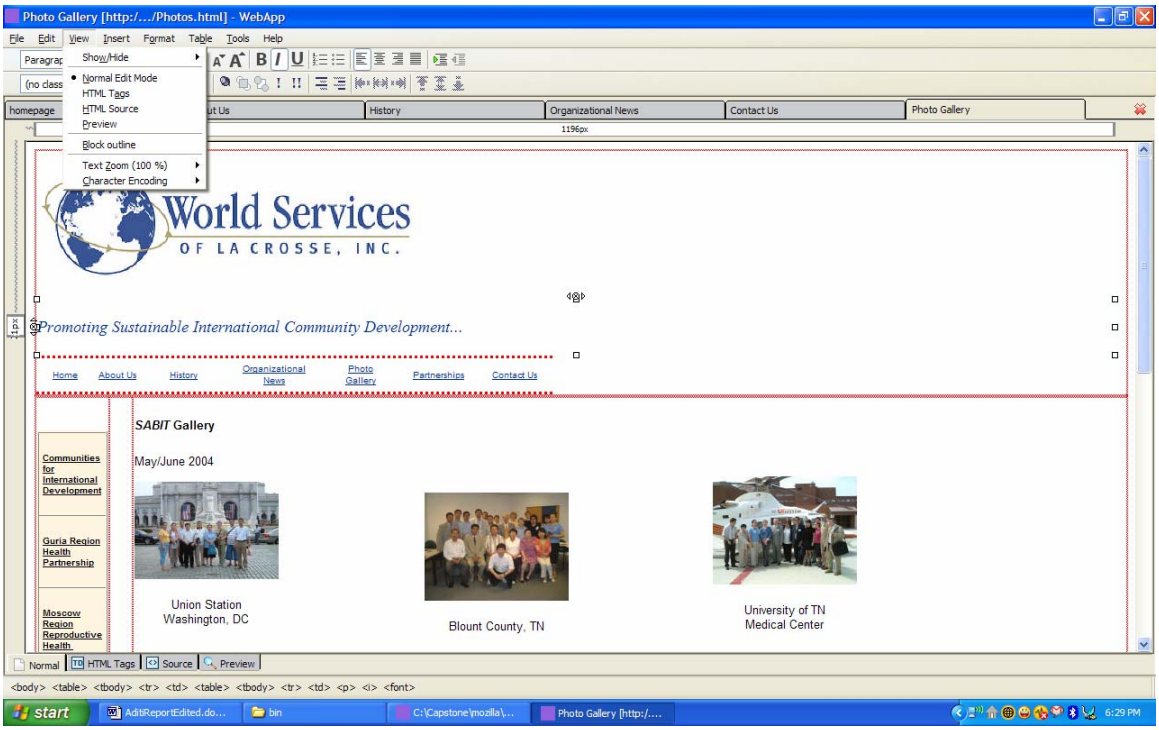
Different tabs view



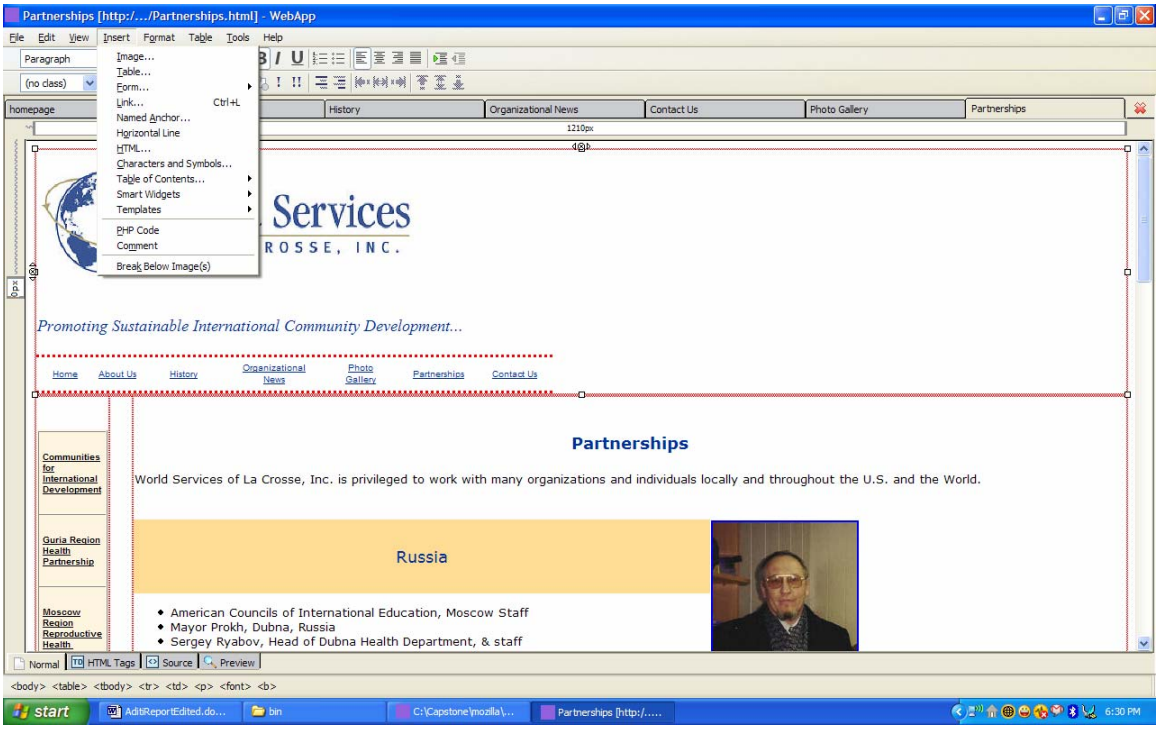
File menu



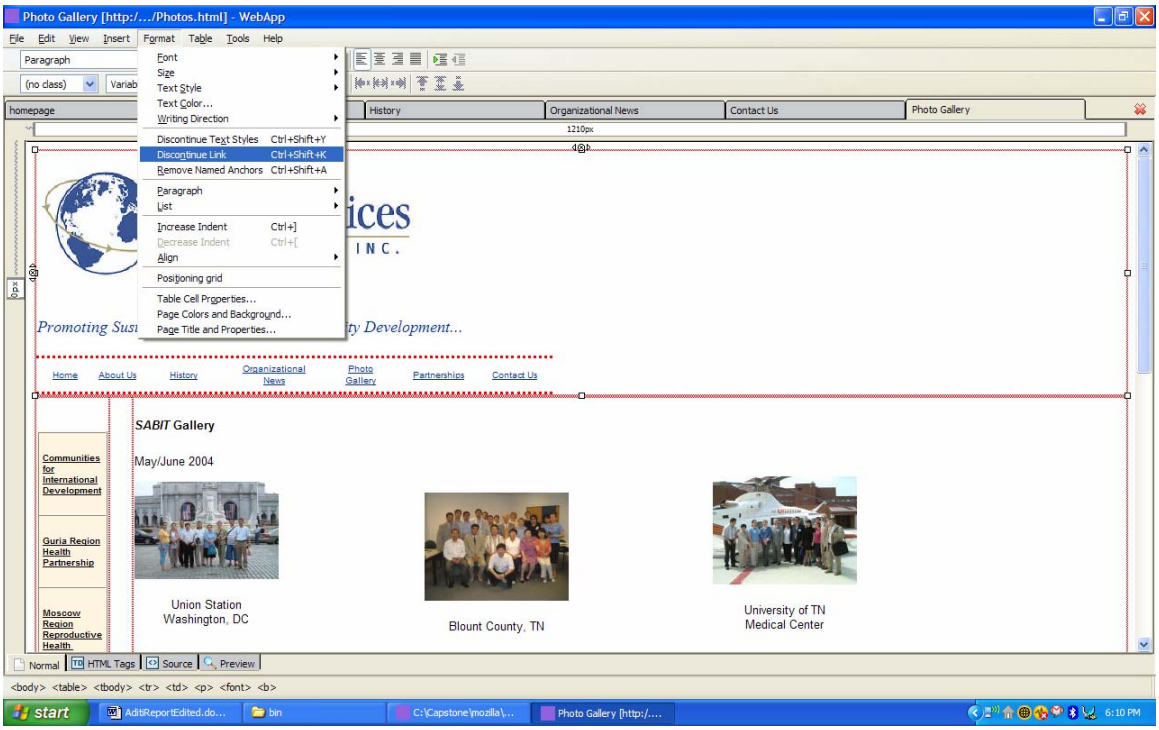
Edit menu



view menu



insert menu



format menu

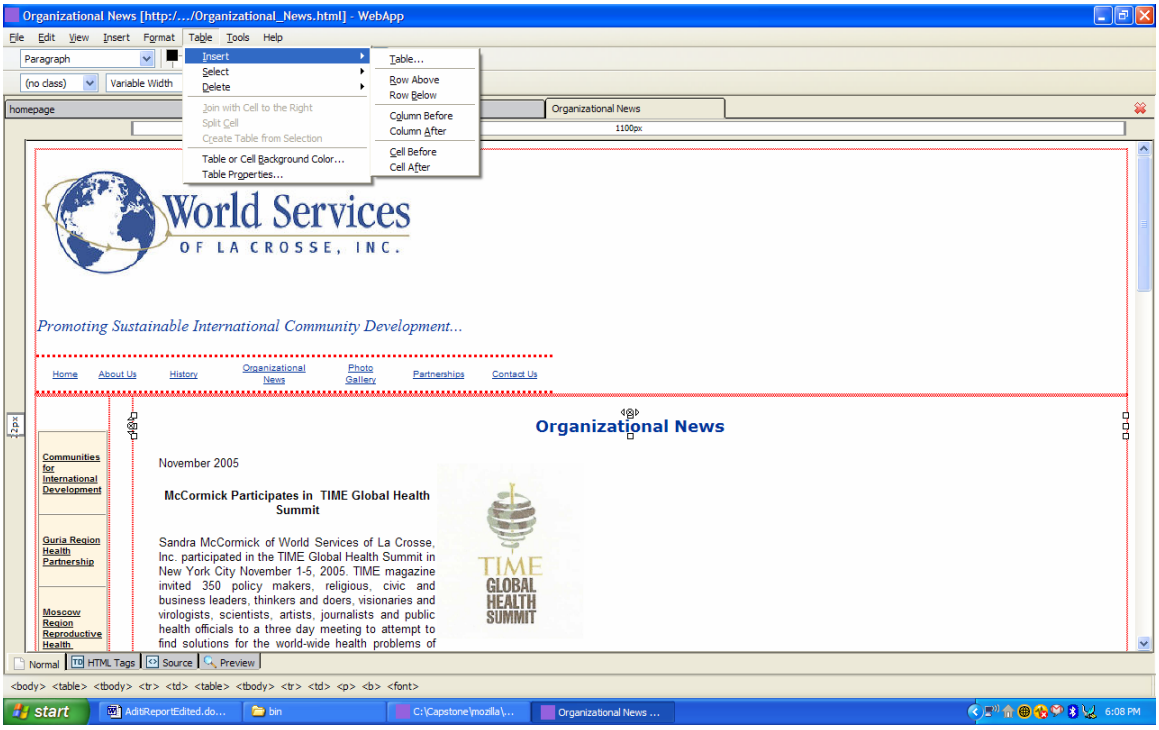
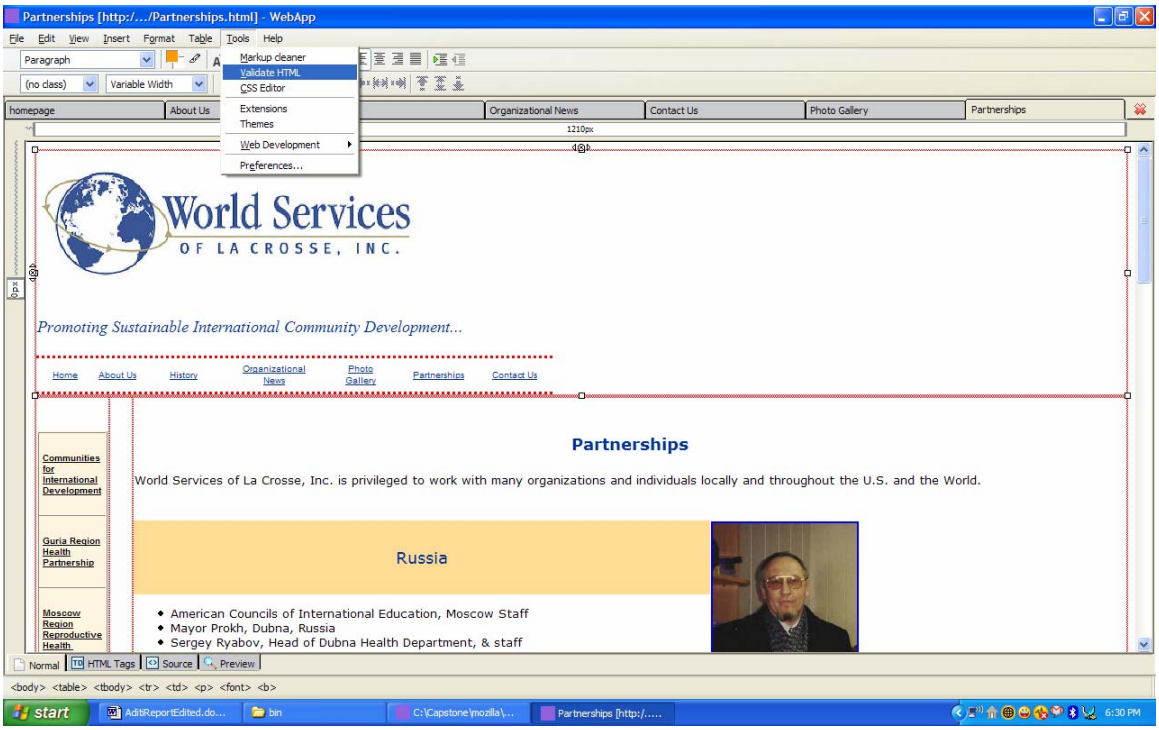


table menu open



tool menu