Online Content Editing - An Evaluation and Comparative Study

Samuel Sambasivam, Ph.D.
Computer Science Department
Azusa Pacific University
Azusa, CA 91702, USA
ssambasivam@apu.edu

David C. Mills
United Kingdom
dave_mills@blueyonder.co.uk

ABSTRACT

Browser Compatibility is quite possibly the most infuriating and frustrating subject for any web developer, this is compounded when the developer is trying to provide support for 'What You See Is What You Get' content editing. This document will discuss the issues surrounding online content editing, web browser compatibility and how they affect the dynamic page builder product that is provided commercially by Dynamic Solutions Development Limited. It is an evaluation of the main product as well as a comparison against other products and solutions that are available today. The original product will be compared against other commercially available products as well as current thinking and theory. Content editing has evolved away from products that must be installed on a machine somewhere in the office and there are now many alternatives. All of these are provided through the use of a web browser, this is largely due to the fact that almost every office and home has a machine with some form of browser and that web browsers are extremely portable. The original product will be evaluated for functionality, compatibility and security and each area will be discussed in detail. The document will address these areas and provide some suggestions as to how they could be improved or in the case where they are not offered provided.

Keywords: Online, Web Browser, Content Editing, Dynamic page

1. Background of the Problem

Dynamic Page Builder is a content editor and website management tool that was written a little over four years ago. The aim of the project was to provide a tool that would allow the user to log in from anywhere they could gain access to a web browser and modify their website through the browser alone. One constraint that still remains is that activeX® can not be used and there should be no need to install any such components on the client machine.

Four years ago content editing was very much the sport of client based applications and the ability to modify a web sites content from within any browser was at very best limited, today however many advances have been made and new browsers introduced. In this paper I intend to address as many aspects of the product as possible looking at how it was originally built what security measures are in place what could be updated and of course I will question whether the original architecture and design are right if the cross browser goal is to be achieved.

There are many solutions available today and I have listed references for these in table 1 (Appendix A) (Other Content Editor's).

2. Dynamic page builder the product

Dynamic Page Builder provides a full web site management tool (through Microsoft Internet Explorer 5.5®© and above) it is far more than a content editor and provides rich features such as the ability to create and modify page description and meta tag information, the ability to modify the page background colour and image along with many other facilities to modify page level information. The product was built with the express intention of making the act of creating and modifying web sites accessible to many different kinds of user. It is possible for someone with very limited knowledge of IT to build a site, just as it is possible for a web master or other IT professional to edit HTML and script through the same interface.

3. HOW WAS THE EDITOR PUT TOGETHER

The editor itself being the core of the system is written as an HTML component, one of the tests I intend to perform in this project is to attempt to remove the Java script from the component and run it independently if this is possible then it is feasible that the editor could be converted to cater for many different browsers as a hook can be written for browser detection then the appropriate code utilised where any major differences occur.

The HTML component has a number of supporting Active Server Pages, these provide the management and administration facilities for the user. I have mapped the applications structure as much as possible in Fig1 below, here you can see the basic flow of the application and how the ASP pages interact with each other. In this section I will discuss each page in turn and detail where any pages have been modified or indeed are completely new compared to the original YUSASP (Advanced Content editor, No date) code. The most significant changes are a number of include files to provide site security and additional functionality these pages are: -

- Commonscript.js
- Logincheck.inc
- ASPupload.asp (ASP Upload, No date)

ASPupload.asp[2] was included as part of the components integration and is based on a sample page provided by Persits Incorporated.

The main editor is produced from the three files that are shown within the HTML component section above (highlighted in green) in Fig1 these are: -

- Ace.htc – The main HTML Component
- Ace.gif – The button image file
- Ace.css – The main style sheet that governs the look of the editor DIV.

We will venture further into the HTML component file later in this document when we look at the code (Figure 1, Appendix A)

The main management page within the site is admin_doclist.asp shown in Fig2 (Appendix A).

A look at the database

Having discussed the user interface and the differences between the original YUSASP[1] product and the Dynamic Solutions product I can see no reason why the database cannot be stored within any ODBC compliant system. Currently the database is in SQL Server format and Fig3 shows a detailed database diagram of the area involved with the editor.

It is clear that the database architecture will need to be completely redesigned in order to cater for many of the changes that I'll propose during this report and it will certainly need to be normalised in order to maintain data integrity. Having analysed the data structure for the site I have extracted only those tables directly related to the editor and its associated customer records (these are necessary for user validation) I have intentionally ignored many of the other tables used for web site management and customer relationship management as they are not relevant.

A look at the code

So what is an HTC?

As mentioned earlier in this document HTC stands for HTML Component, this technology

was introduced by Microsoft® with Internet Explorer 5® and is NOT backward compatible. This presents many issues for any developer that chooses to implement this approach, on the one hand it is extremely powerful and many applications are adopting its use within the corporate arena, whilst on the other it rules out the use of any other web browser as the technology only functions with Internet Explorer 5 and above. I have provided a link to the overview for HTC (HTML Components, No date) for those that wish to investigate this area further, essentially HTML Components allow developers to create re-usable code within their web applications. For the most part HTC and ASP / ASPX seem to go hand in hand, and this is indicative within large corporate environments that utilise Microsoft® software throughout.

An HTC file is simply an HTML document with some additional tags. The additional tags allow the component to react to events that occur within the parent document and expose properties and methods in much the same way as any traditional component written in Visual Basic or C++. Below I have included the window_resize() code that will react to the resize event of the parent window.

```
function window_resize() {
                var          test          =
        "idContent.editorWidth               =
        document.body.clientWidth - 20;"
                test                        +=
        "idContent.editorHeight              =
        document.body.clientHeight - 200;"
         if (window.onresize) {
        window.setTimeout(test,250);
        }
}
```

The <public:Attach event code of the HTC is used to point to the above script, whilst an investigation into the uses of the HTC would be extremely interesting it is important to investigate its removal from the underlying architecture as this removes the browser restriction. The code seen above is simply a standard JavaScript that can be found in any client side script within an HTML page.

In the head of the page an XML namespace is declared as follows: -

```
<html xmlns:ACE="ace.htc"
 xmlns=
"http://www.w3.org/Profiles/xhtml1-
transitional">
<head>
```

Browser Compatibility

Here is where the majority of the problems lie, whilst it is a relatively simple task to remove the HTML Component from the code the issue of browser compatibility will remain. The problem is largely due to the fact that the DOM (Document Object Model, No date) used by Microsoft® is not compatible with the DOM used by Mozilla®, whilst both follow the W3C standards for standard browsing neither support editing functions in the same way. Later in the document I will illustrate the clear differences between the two approaches.

System Compatibility and Security

There is a great deal of embedded SQL used within the site and this really should be placed in stored procedures. The site security is generally quite poor and needs to be addressed, though most of my observations in this area fall outside the scope of the project it will be affected by a number of them. Embedded SQL, Session security, and stored procedures are the most pronounced examples.

When looking at the three tables in Fig3 it is clear that they will need to be normalised furthermore the documents table in particular needs a great deal of attention as it ideally needs to store all the possible properties of the head and body tags of a standard page or store the whole document in one column.

4. An Alternative Approach

Throughout this chapter we will discuss a new security model in order to protect the user and their data, how ASP .Net can be used to improve the functionality of the editor, the differences between the IE® and Mozilla® DOM, how this affects online content editing and why I believe that the HTC can be removed from the current Dynamic Page Builder product.

The Security Model

I have decided to implement a new .Net based solution in order to provide a higher level of security. I have included a page access string along with encrypted sign in and password storage. The architecture I have used is based on an existing Visual Basic COM component. It has been re-written using .Net It is important to note that the concept is based on a component originally written by Blue Sands Inc. (Blue Sands Inc., No date).

The component has had the following additional features included, extending the original COM component a great deal.

- Page Access String
- Cross server session management (Can now cope with clustered servers)
- Encryption (Basic but sufficient)
- User identification
- Sign In Log (detailed information of when users Sign In and Out)

To an extent this is tied to code that I have written specifically for this project. However, the security model I have created can easily be applied to many sites. I chose to implement this approach as whilst I appreciate that ASP .Net now offers session management through SQL Server I wanted to have a great deal more control over what was being stored and how.

In figure 4 (Appendix A) shows the basic flow that the user will experience when logging onto the site for the first time.

In figure 5 (Appendix A) we have shown a more detailed flow of what is actually going on when a user visits the site, as I will refer to this figure several times.

At this point we would like to highlight some of the unique features of this session management approach: -

- As a central database is utilised to store session information the architecture allows the use of clustered servers, in fact any individual site could be hosted on several servers. Perhaps streamed data being stored with one host whilst the main site is hosted elsewhere and furthermore secure servers could be at yet another site. The fact that different sections of the site can be stored on separate servers is a major advantage of this approach.
- As the session time out is managed within the context of the component it can be set to absolutely ANY reasonable time scale up to the maximum amount of time that a cookie can be stored on a users machine and as little as a few seconds. Of course the later is impractical but it might be the case where sensitive information is to be viewed then we might set the timeout to a minute or 45 seconds.
- Due to the way the timeout is handled another really nice feature is that the user can completely close their browser window or even reboot their machine. When they come back to the site they will still be logged in provided that the session has not timed out (I will explain how this is achieved in detail later in this section).

Once it has established whether the user is signed in or not, either a message is displayed to inform the user they are not signed in or the welcome message is displayed.

A number of further checks are performed and these follow, as you can see in figure 5 if this is the first time a user has visited the site and they are not a member they will simply see the default page with the Sign In dialogue box, they will also receive the default menu (which is not clear from the diagram). If an attempt is made to login without a valid username they will simply receive a message informing them that the login attempt has failed, equally if a valid username is used and the password is incorrect then the user will receive the same message. As shown in the diagram there is an additional step behind the scenes that actually updates the user table keeping a count of attempted logins, if this is greater than or equal to three then the user will be locked out of the database and will need to contact a member of support (in this case me) in order to be able to log back into the site.

Assuming that the user enters a correct username and password the process then retrieves all relevant data from the database regarding the user and uses this to create a number of rows in tbl_ssn_sessionData in

order that they can be utilised throughout the rest of the site. During this process a cookie is created on the users machine and an encrypted session value is written to the cookie.

From this point on any page that is accessed by the user will call the session management component. It will check that they are entitled to view the page they are requesting before it is rendered to the browser, this way the end user will never be able to view a page that they do not have access to.

A Closer look at the session manager

In this sub section we will take a closer look at the session manager component and how it functions. Figure 6 (Appendix A) shows how the functionality is encapsulated within the component in order that no database connections are made directly from the page at any point, it will also show the methods that are exposed by the component.

5. The DOM Compared

The title of this section is perhaps a little misleading; it would imply that there is only one document object model and perhaps W3C would like us to adhere to the principals they have lain down. There are still a number of discrepancies and of these one that stands out is the "contenteditable" property only available within Internet Explorer 5.5® and above. This is an extremely powerful attribute and provides the ability to make DIV's, SPAN's and iFRAMES editable. An example of this can be found at http://www.dynamicpagebuilder.co.uk/editorpage.aspx (please bear in mind that you will need to sign in to view the page due to the security that has been implemented within the site. If you need a username and password for this purpose please contact me.)

The W3C version of the Document Object Model is currently at Core level 3. Microsoft® however extended the DOM from core level 1 and amongst the extended features are those that allow advanced editing. Netscape whilst far more compliant to the official DOM have extensions of their own. A curious fact is that both Microsoft® (W3C DOM, No date) and Mozilla® (Mozilla® DOM, No date) are way behind W3C (W3C DOM , No date). Microsoft® only support W3C Core level 1 and Mozilla® 2, when both vendors catch up to level 3 this discussion may well be irrelevant as provision for online browser editing will be placed into the latest DOM. Details of this can be found at (Microsoft DOM, No date).

I do not intend to discuss the many issues that surround vendors building their own proprietary versions to suit their own needs, in an ideal world each vendor would conform to a common standard but I believe that we are a number of years away from that becoming a reality. So though this section of the document is short, it serves to highlight the fact that there is a common standard managed by W3C and that none of the browser vendors currently meet this. In turn, this means that a true cross browser editor can only be achieved through compiled code. For a project such as Dynamic Page Builder this would involve a complete re-write or the purchase of third party code. As these are not options within the original remit the goal becomes a slightly different one, we need to find a solution that gives the end user the impression that they are looking at a seamless cross browser product.

With this in mind I have produced some test pages on the website http://www.dynamicpagebuilder.co.uk and links can be found on the Project Tests Page. In the following sections I will discuss the .Net Framework and its benefits for this project.

A brief look at the .Net framework

At this point it seems appropriate to discuss the reasons for my choice of Microsoft® .Net Architecture. I have not chosen it simply because it's the latest buzzword and agree that there are PHP solutions that offer the full cross browser package. One of the upsides of the .Net framework is the fact that it has truly embraced web services. This is an area that I intend to discuss in this chapter and will cover in more detail later in the XML and web services section. I found following diagram (Web Services Explained,

No date) (Appendix A) which whilst simple is effective in describing how a web service functions.

Without delving too deeply into web services, it is clear that the component used on editorpage_V003.aspx on the dynamic page builder web site could easily be amended expose its properties and methods over the internet, I have not included the component as a web service, due to time constraints and the fact that I would prefer not to make it freely available for use just yet. However, the possibilities for building the project as a web service are certainly there and I will include it in the further investigation section at the end of this document.

Using ASPX

One of the great strength's of ASPX is the fact that the client and server side code are very neatly separated into their respective areas. Client side code existing within the context of the ASPX and server side code in the VB, of course there is some crossover here. It is possible to place server side code within the ASPX page as it used to be accomplished with ASP; however the facility is now available to neatly split the two, so the business layer and presentation layer remain separate.

Utilising the power of the code behind page has meant that I have been able to establish which browser the user has in the middle tier and only render the relevant content dependant on the test. This in turn means that from a users' point of view they only see JavaScript that is relevant to their particular flavour of browser.

Using HTML Components (HTC)

As the HTC provides a way for developers to extend Internet Explorer it was heavily used in the original product produced by Dynamic Solutions Development, however its use is a restriction in itself. While I agree that it is convenient to be able to create properties and methods within an HTML document, in order to expose DHTML behaviours. I have always maintained that there must be a better and more standards based solution.

XML and Web Services

One of the great benefits of a web service is the fact that it allows data to be passed around the internet seamlessly from one system to another and one browser to another without the need to worry whether either is compatible. A quote from the ASP .Net Kick Start guide [7] reads "If the browser can read a web page it can use a web service, regardless of the operating system" this is a boon for a project such as this.

Sadly I had not scoped the project to cater for this and in order to meet my deadline I cannot investigate web services as a means of producing an online content editing system. I will most certainly be including it in the further investigation section and have made reference to it here as I would like you the reader to gain a better understanding of my thoughts and where the project will go once this phase is over.

6. Project Tests and Solutions

In this section, I will discuss the solution I have provided on the project website http://www.dynamicpagebuilder.co.uk I will look in detail at the program flow and how I have achieved the cross browser result displayed within the editorPageV003.aspx page. As a reader of this document I strongly recommend that you view this page mentioned above in at least Internet Explorer®, Netscape® and the FireFox® browser's. Below in figure 8 (appendix A) we have placed a diagram of the basic flow that occurs when the page is opened. Later in this section I will discuss each phase in turn and explain what is happening in detail. One of the underlying drivers for this code is to minimise the need to post data back to the web server in order to maintain performance. Postbacks can totally sap the performance of an application if not handled carefully.

When the page is initially called on the server the code behind (actually it is now the project assembly dll located in the bin folder on the web server) calls the Session

Manager component, this will do one of two things

a) It will return the user to the sites home page
b) It will return details of the user and allow access to the page.

In the first instance, session manager has determined that the user is either not logged on. Or that they do not have access to the page, either way they will be returned to the home page.

In the second, (and I would hope more common instance) the user will see the editor page. However, I am jumping ahead slightly. Before the user gets to see the page an awful lot happens, first of all, the page will call an embedded 'User control', for example editorpage.aspx calls dpbEditor.ascx it is within the control that the majority of the work is done. The control will then check the type of browser the user has and again at the client (just in case there is a discrepancy with the browser type returned from session manager). Secondly the configuration file is read, the pages controls are drawn from the ASPX part of the page after which the relevant scripts are drawn from the VB side of the page. Once all of these phases have competed successfully the page is rendered to the client.

Once these things have happened, the code is rendered to the browser and the user will see the editor as in figure 9 (Appendix A).

Figure 10 (Appendix A) below shows the actual construction of the ASP .Net pages and related files and how they interact. I have localised this diagram to the editorpagev003.aspx as this is the final fully functional version.

It is clear from the diagram that all ASP .Net pages are made up of three physical files that are referenced as one logical one, the ASCX extension simply indicates that this page is a control otherwise it can be thought of much like a normal ASPX page. The RESX extension indicates the resource file, this is a file used by the framework in order to locate where the relevant resources are for the page.

The diagram depicts the physical files. However, if we were to look at the logical mapping we would simply see the editor page and the commands.xml page as in figure 11 (Appendix A).

We spent a great deal of time when considering the design of the editor container and made sure that wherever possible. I used tags that were supported by both mozilla® browsers and Internet Explorer®. This has meant that whether viewed from FireFox®, Netscape or Internet Explorer® the results are similar, there are few idiosyncrasies but to the untrained eye it looks the same. From here I had to determine what really makes the whole thing tick and not surprisingly it comes down to some clever code provided by the vendors the Key to all of this are two attributes: -

a) designMode
b) contenteditable

The first of these attributes is available in both browsers and essentially switches the content of an iFrame, DIV, SPAN into an editable region of the page. However, there are differences depending on the browser. The second is only available through IE®.

The major difference between the Mozilla® offering and the Microsoft® offering lies here, both vendors make allowance for a page to be switched into editable mode, however only the Microsoft® offering allows for individual elements within that page to be switched on or off dependant of the value of the content editable attribute. A detailed discussion can be found at (Microsoft® MSHTML , No date) Microsoft's® MSHTML page.

The next important function is how to actually modify the HTML in order to produce effects like emboldening or underlining text, this turns out to be very easy, it is simply a case of making a call to the execCommand function supported by both browsers albeit slightly differently. I have found that calling execCommand with all three parameters (as detailed below) functions in both of the core browsers with no adverse effects.

execCommand("[Command Name]", [User Interface], [Value])

Where Command Name would be Bold or underline etc, User Interface would be true or false and Value would be present for any command that needs it, such as forecolor or null if not.

An interesting error came out of testing, and that was the fact that the User Interface parameter must always be set to false when calling the Mozilla® version or an error is returned, whereas the Microsoft® offering handles this well whether it can display a UI or not.

7. Final Report

Online content editing a comparison

Of the online content editors available, there are only three base solutions from which they are derived: -

1 – Internet Explorer (content editable elements and editable page supported.)
2 – Mozilla (editable page supported)
3 – PHP compiled cross browser compatible code base

Whilst the PHP solutions are very interesting, they fall outside of the scope of this project. They do however, warrant further investigation at a convenient time.

Of the projects I have reviewed during this evaluation, none have stood out as being capable of fully replacing client side applications such as Microsoft FrontPage®©, Macromedia DreamWeaver®©, or Microsoft Visual Studio®©. However, I am certain that further investigation in the area and the advances that are being made with broadband technology, will improve the features of such products and it will not be very long before professional developers begin to use the online tools in place of the more traditional client side packages.

Summary and Conclusions

I have seen some very impressive solutions and some very poor ones, however as I have already mentioned all of those reviewed are derived from one of three sources and I am certain that the proposed solution I have provided at http://www.dynamicpagebuilder.co.uk/editorpagev003.aspx is one of the only true offerings that address the discrepancies between the variations of the DOM. From an academic point of view I will continue to pursue this project in an effort to produce a true cross browser compatible online editor, however from a commercial point of view I believe for any company providing a service or selling a product the fact that Internet Explorer has over 90% of the market (Browser Statistics, No date) cannot be ignored.

Internet Explorers® dominance of the particular area remains, at least for now.

Further Investigation

There are a number of areas that warrant further investigation and I have provided a simple list below:-

- PHP compiled code solutions
- Web Services and the use of SOAP (Simple Object Application Protocol)
- Third party FTP solutions should be considered
- Future browser releases and what support they will provide for online content editing.

REFERENCES

Advanced Content editor
http://www.yusasp.com
ASP Upload http://www.persits.com
HTML Components
http://msdn.microsoft.com/library/default.asp?url=/workshop/author/behaviors/howto/creating.asp
Blue Sands Inc. http://www.bluesands.com/
Document Object Model
W3C DOM
http://www.w3.org/DOM/
Microsoft® DOM
http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dom/domoverview.asp
Mozilla® DOM
http://www.mozilla.org/docs/dom/
WYSIWYG Pro
Producthttp://www.wysiwygpro.com/demos/demo2.php
Web Services Explained http://www.service-architecture.com/web-services/articles/web_services_explained.html

Microsoft® MSHTML
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmshtml/html/mshtmleditplatf.asp
Etive Web Controls
http://www.etive.com/software/dotEtiveFTP/
Browser Statistics
http://www.w3schools.com/browsers/browsers_stats.asp

APPENDIX A

Table 1 Other Content Editor's

IE 5.5 and above Only
http://www.interactivetools.com/products/htmlarea/
http://www.cutesoft.net/ASP.NET+WYSIWYG+Editor/Requirements/default.aspx
http://www.blueshoes.org/en/javascript/editor/
http://sourceforge.net/projects/bpeditor/
http://www.xtort.net/xtort/protopad.php
Mozilla Only
http://www.bitfluxeditor.org/
http://composite.mozdev.org/
http://www.mozilla.org/editor/midas-spec.html
http://mozile.mozdev.org/
Cross Browser
http://www.wysiwygpro.com/demos/demo2.php
http://www.hardcoreinternet.co.uk/
http://www.phpwcms.de/index.php?id=3,0,0,1,0,0
http://www.kevinroth.com/rte/demo.htm
http://vietdev.sourceforge.net/portal/html/index.php

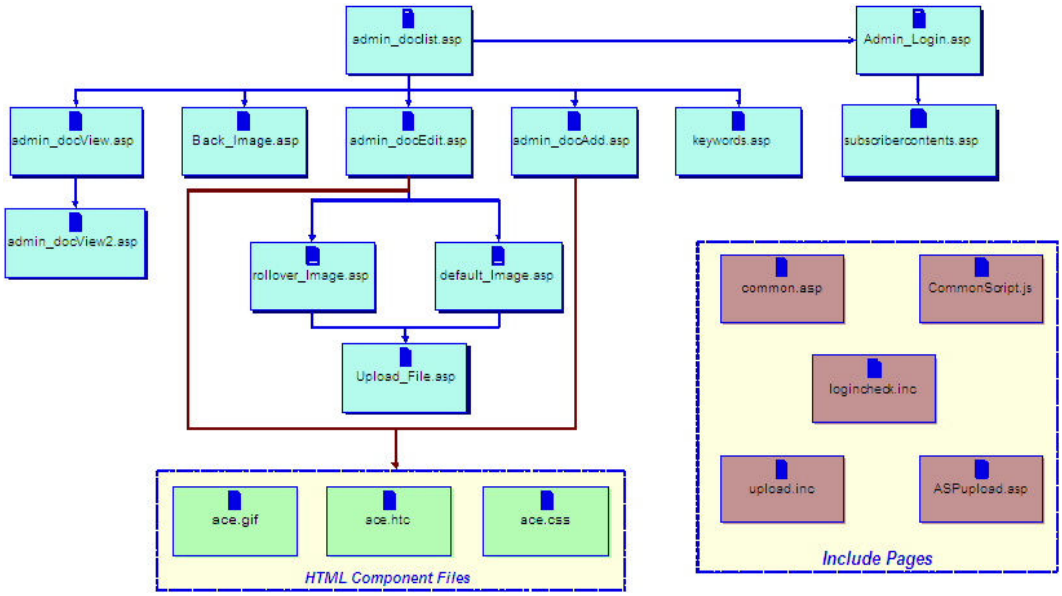
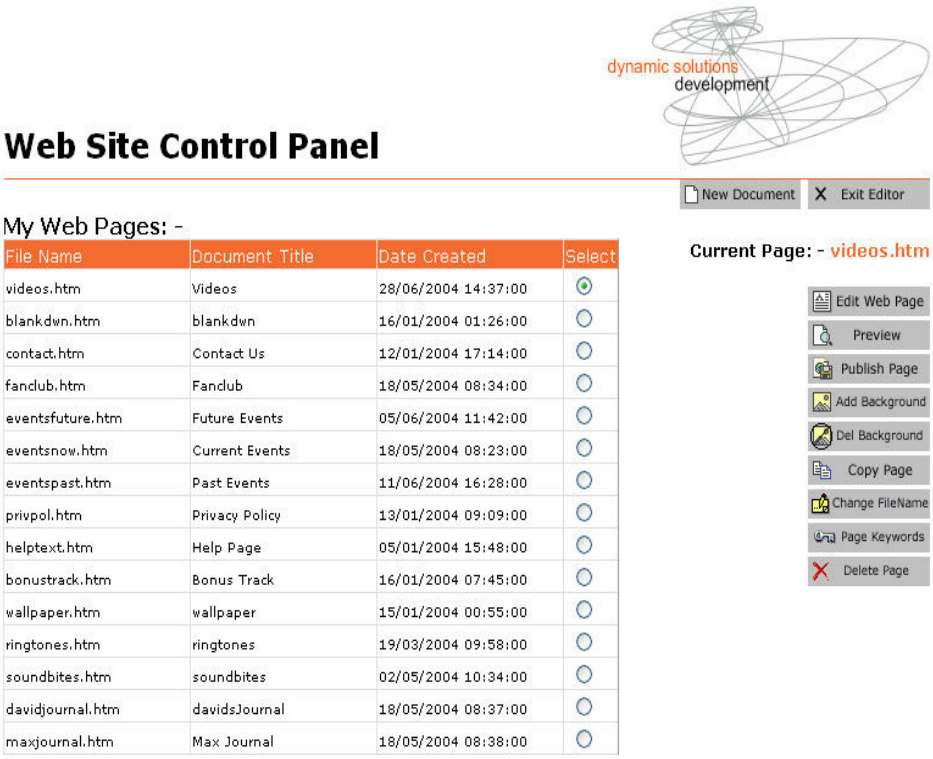


Figure 1 Content Editor Files and interactions



File Name	Document Title	Date Created	Select
videos.htm	Videos	28/06/2004 14:37:00	◉
blankdwn.htm	blankdwn	16/01/2004 01:26:00	○
contact.htm	Contact Us	12/01/2004 17:14:00	○
fanclub.htm	Fanclub	18/05/2004 08:34:00	○
eventsfuture.htm	Future Events	05/06/2004 11:42:00	○
eventsnow.htm	Current Events	18/05/2004 08:23:00	○
eventspast.htm	Past Events	11/06/2004 16:28:00	○
privpol.htm	Privacy Policy	13/01/2004 09:09:00	○
helptext.htm	Help Page	05/01/2004 15:48:00	○
bonustrack.htm	Bonus Track	16/01/2004 07:45:00	○
wallpaper.htm	wallpaper	15/01/2004 00:55:00	○
ringtones.htm	ringtones	19/03/2004 09:58:00	○
soundbites.htm	soundbites	02/05/2004 10:34:00	○
davidjournal.htm	davidsJournal	18/05/2004 08:37:00	○
maxjournal.htm	Max Journal	18/05/2004 08:38:00	○

Figure 2 Main administration control panel

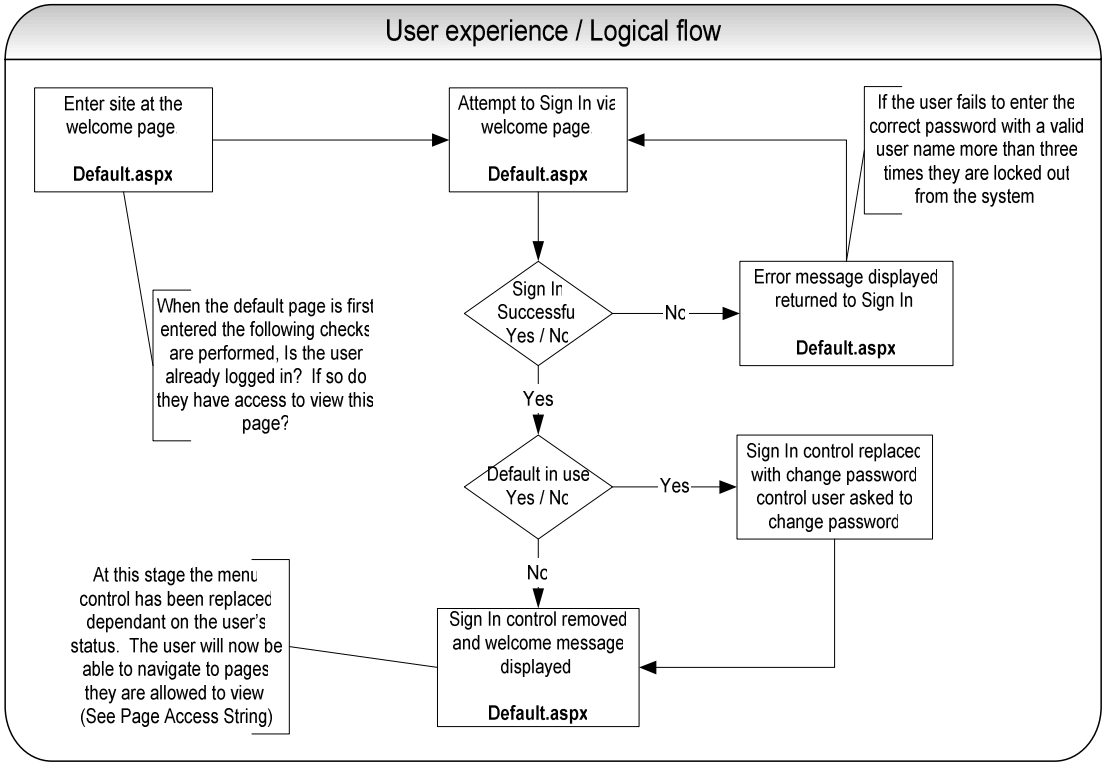


Figure 3 User experience and logical flow

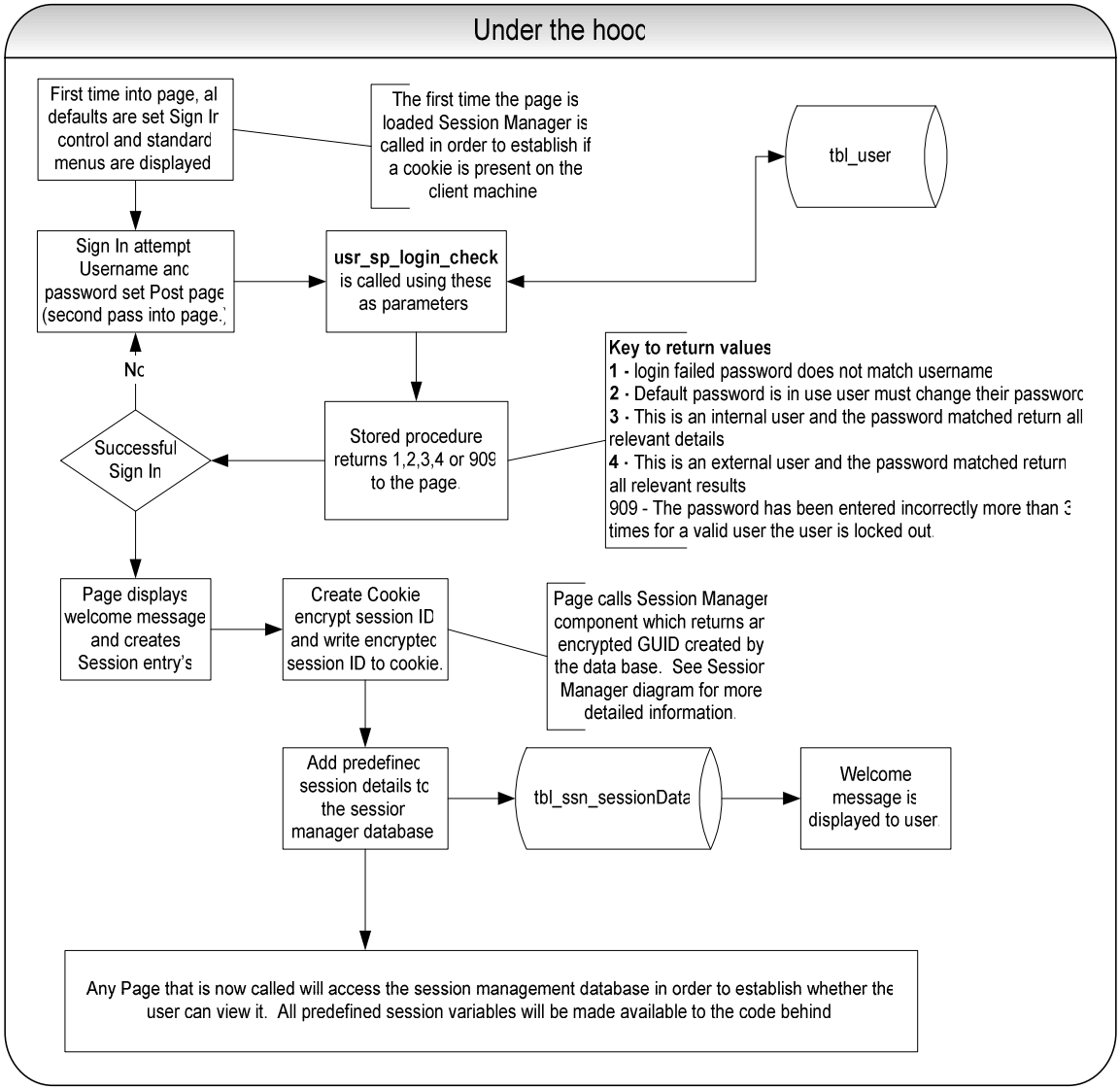


Figure 4 under the hood

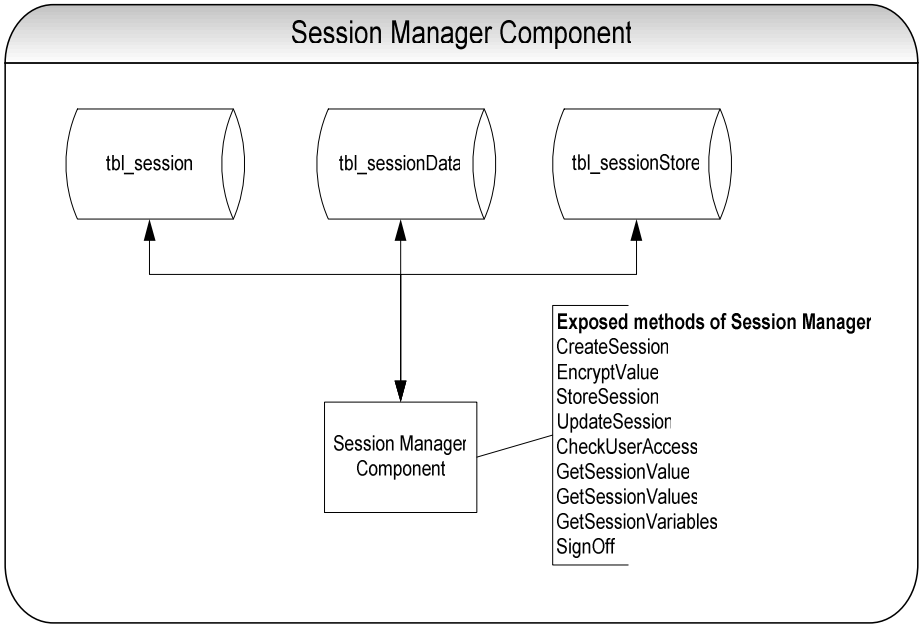


Figure 5 Session Manager Component

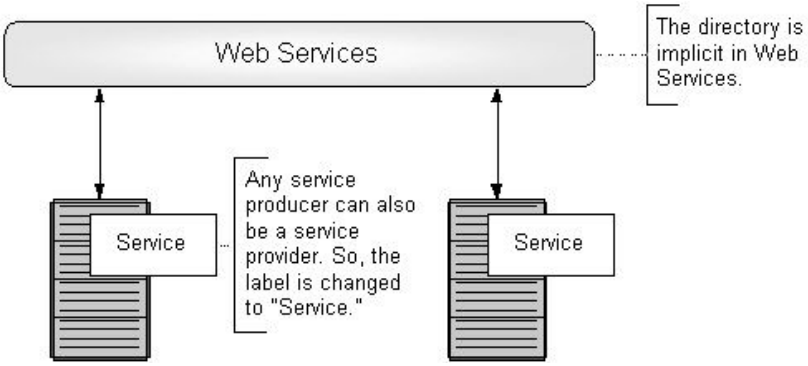


Figure 6 Web Services

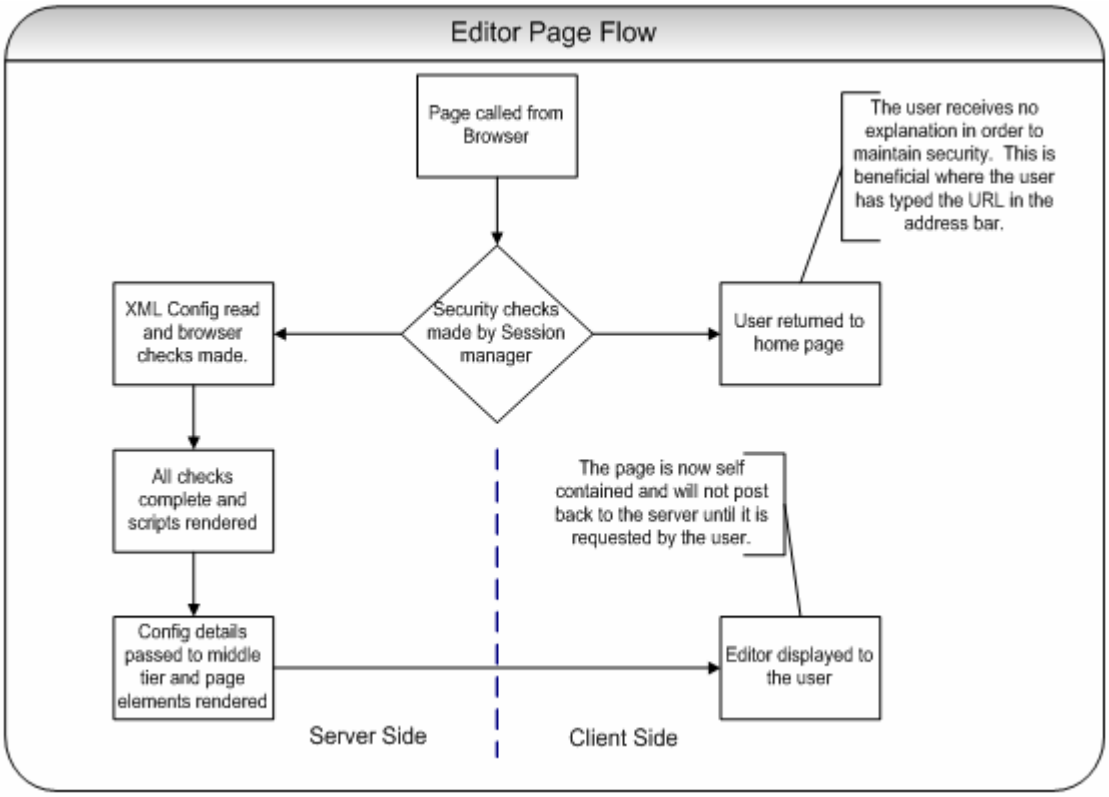


Figure 7 The editor Page Flow

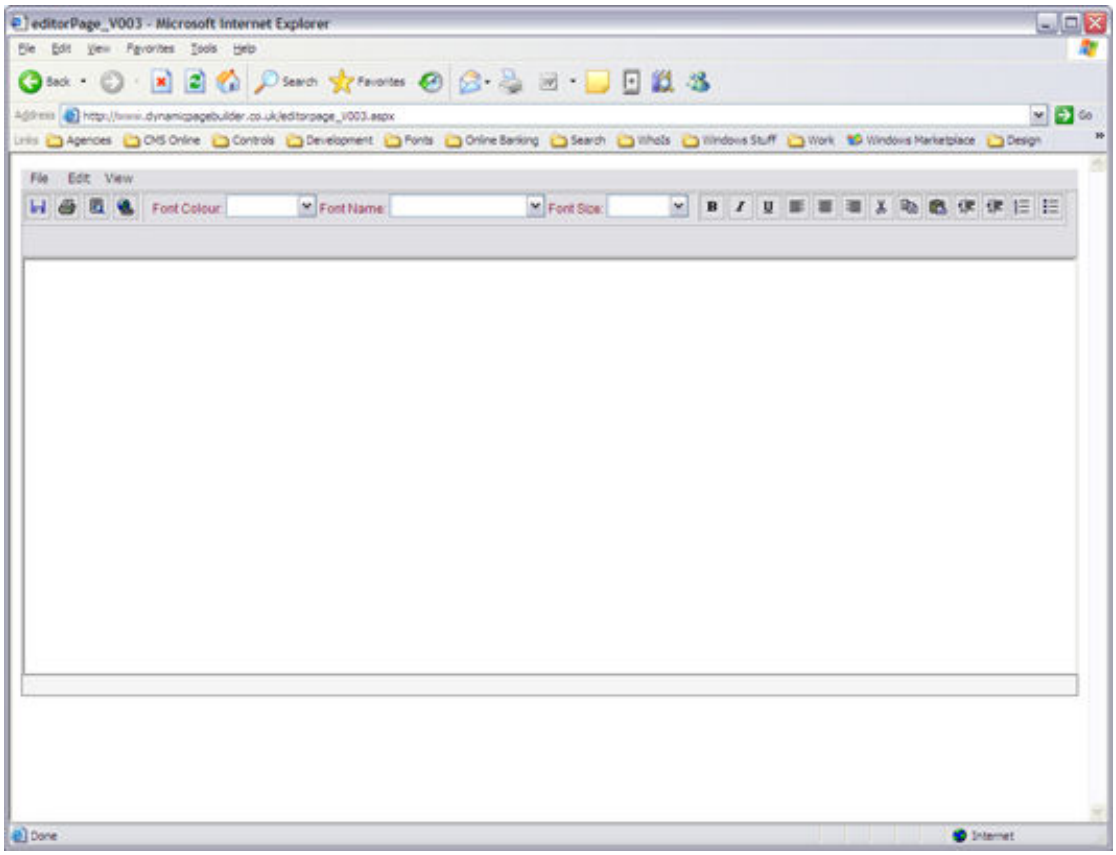


Figure 8 The New Editor

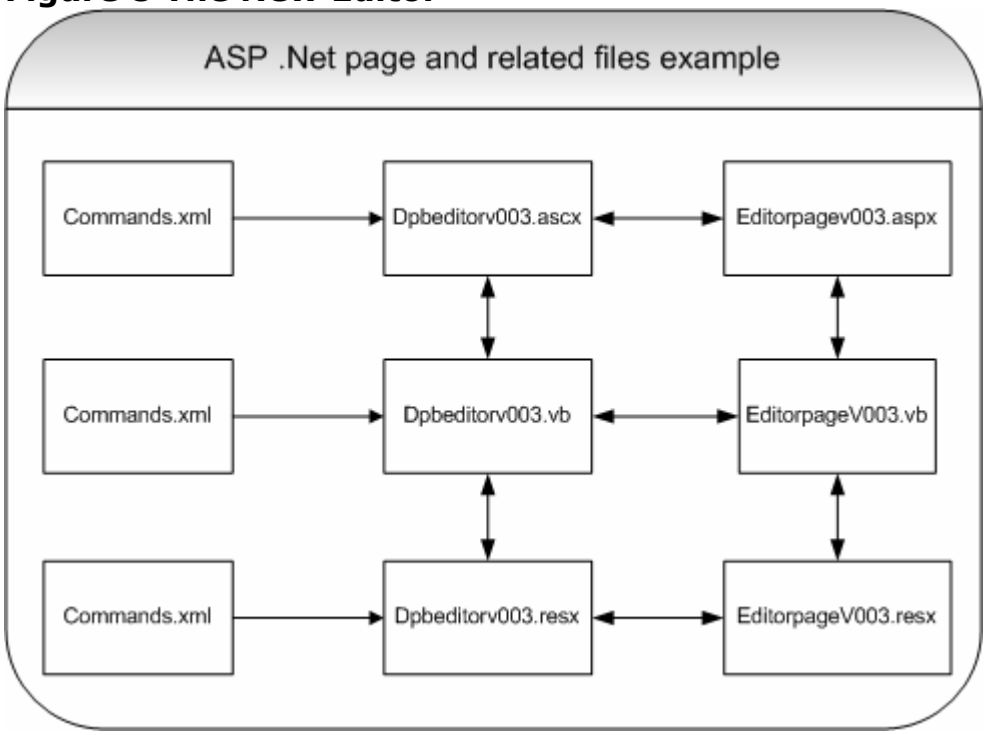


Figure 9 ASP .Net Page and related files

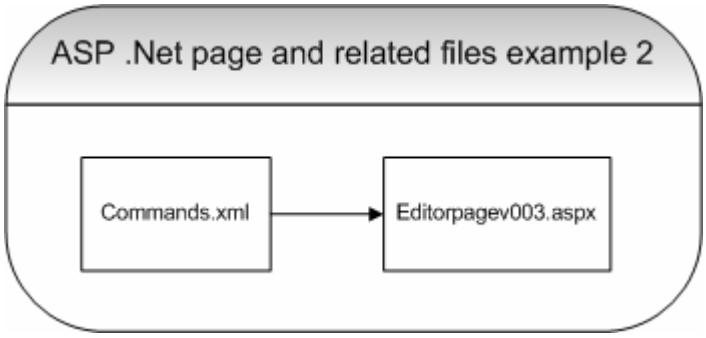


Figure 10 Virtual impression of ASP files