***Coursework 1***

**Rapid Application Development**

(Web Browser)

***F21SC: Systems Programming and Scripting***

*Submitted by*

**Ronny George Mathew,**

***MSc. Computer Systems Management,* 2012**

**Table of Contents**

1. Introduction ……………………………………………… 2
2. Design Considerations ……………………………………. 2
3. Requirements Checklist …….…...………………………... 3
4. User Guide ……………………………………………….. 4
5. Developer’s Guide ………………………………………… 9
6. Testing …………………………………………………… 20
7. Conclusion ……………………………………………….. 27
8. References ………………………………………………… 28

**1. Introduction**

The aim of this project is to design a web browser application using Microsoft Visual Studio 2010. This task has to be accomplished without using the C# web browser class and by using only the installed libraries in Visual Studio for C#.

This project has helped me to get familiarized with visual studio and the various tools and controls associated with it. It made me realize the ease of coding with C# and visual studio.

I have designed my browser around the following Assumptions and limitations:

* The main content of the browser is HTML text. Clicking on the render button will render the required HTML page.
* There is no separate button to view source as the main content in this browser is source. Clicking on the refresh button after rendering will load the source code.
* Only the URL of the visited webpage is stored in the history.
* The favorites’ names are just one word, i.e. It does not include spaces.
* I have not used the web browser control for anything else other than rendering.
* Creation of a new tab is done automatically when you click browse.

**What is a web browser?**

A web browser is an application for retrieving, presenting and traversing information resources on the World Wide Web. An information resource is identified by a *Uniform Resource Identifier* (URI) and it can be a webpage, image, video or other content. A web browser can also be defined as an application software designed to enable users to access, retrieve and view documents and other resources on the internet. Some of the major Web browsers are Chrome, Firefox, Safari, Internet Explorer and Opera.

**2. Design Considerations**

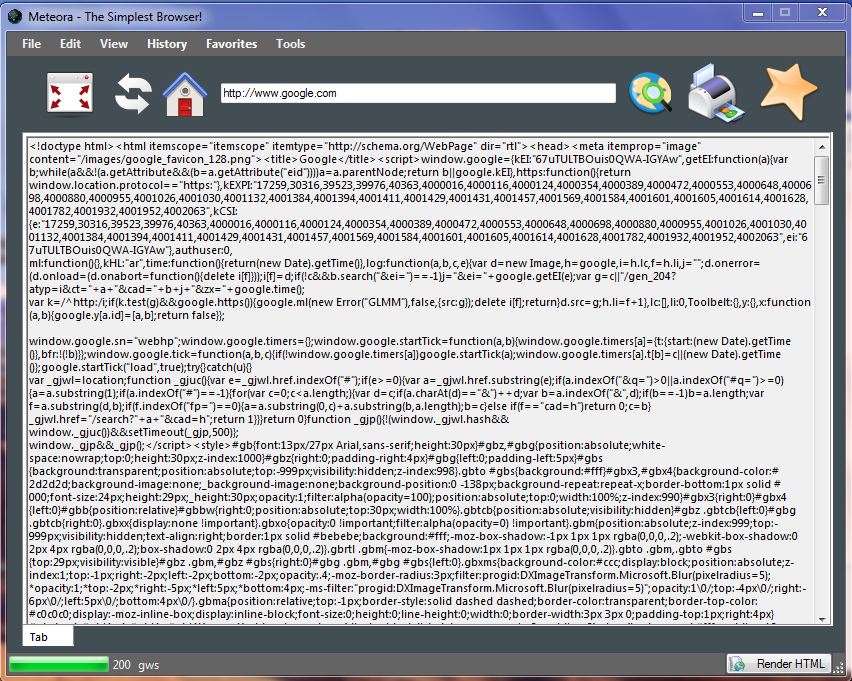
The design of this browser aims at simplicity and ease of use for a user. A User should not go searching for the required buttons under various menus. For this purpose, most of the required options are provided in the form of buttons directly on the main UI.

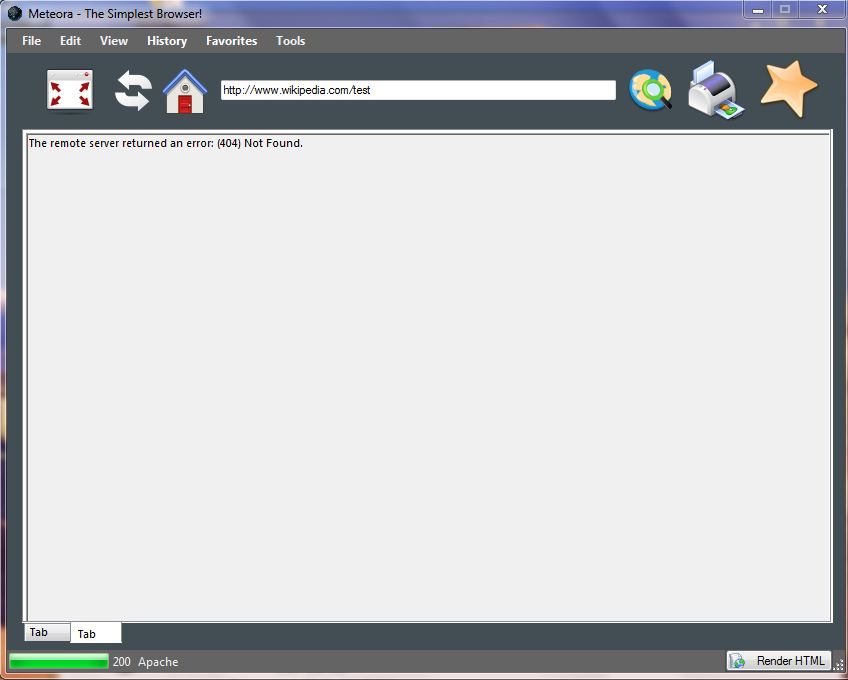
Each new query will have its own tab. Users need not create a new tab every time they need to visit a new page, they can directly type the address and click browse, without worrying about losing any other page.

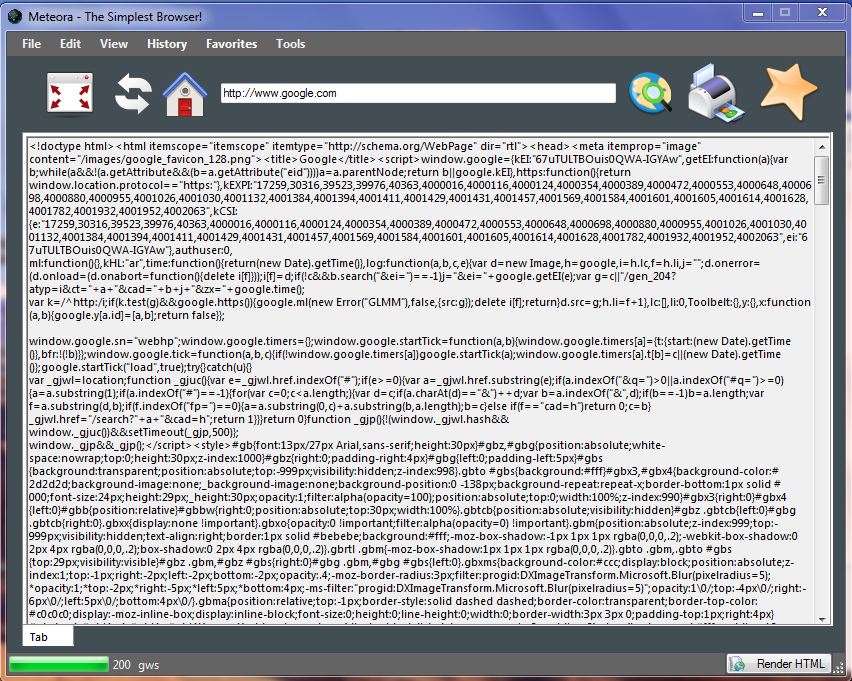
I have used basic multithreading by using a new thread for each tab and for some other jobs. There are also 3 background workers running together which helps the application to function seamlessly.

**3. Requirements’ checklist**

All the requirements for this project are met either fully or partially. Now, I will detail on the completion of each requirement.

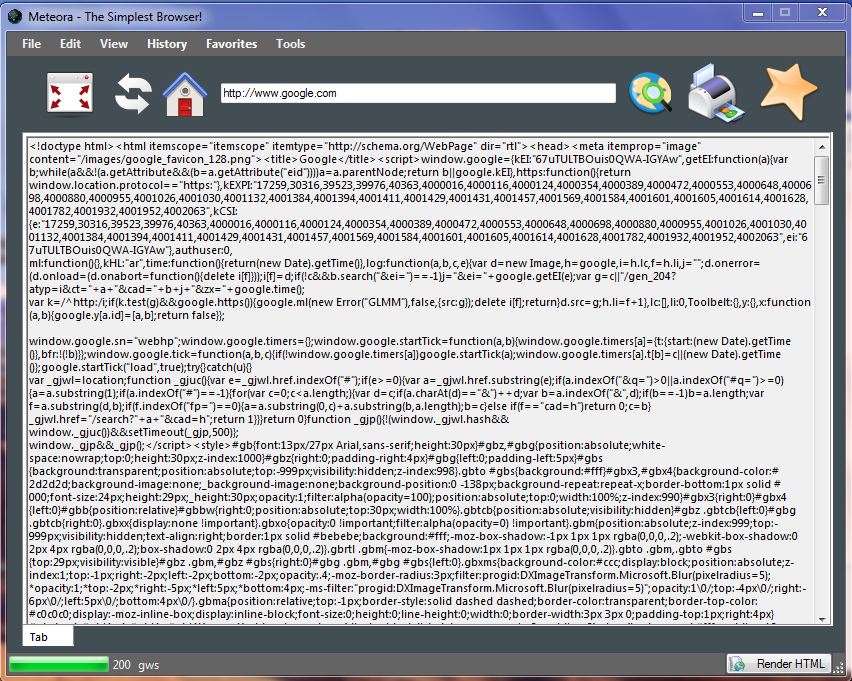
**Sending HTTP request messages:** The URLs typed inside the address box will send HTTP request messages and load the contents.

**Receiving HTTP response messages:** The HTTP response will be displayed in the tab if there are any errors. The status code for last request will be displayed on the status bar on the bottom along with the server information.

**Home page:** A home page button has been provided. Clicking on the button will take the user to the home page. User also has the ability to change the homepage.

**Favorites:** Users have complete control over favorites through the favorites manager. A user can add, edit or remove any of the favorites. It is also automatically loaded on startup to the favorites menu in the menu bar.

**History:** The URL of the pages visited will be stored in the history. The history items also will be automatically loaded to the history menu on startup of the application and is constantly updated.

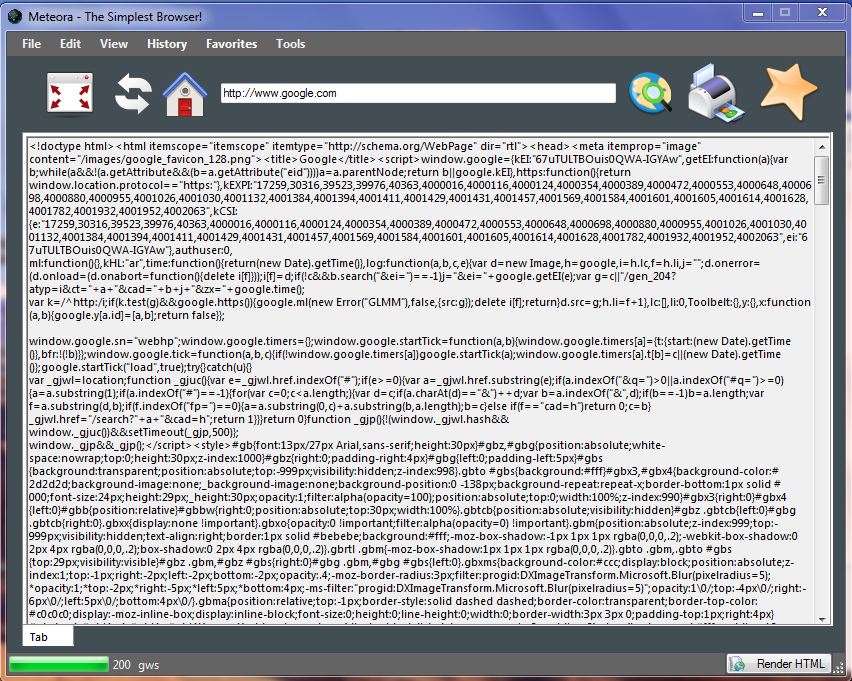


**Printing the content:** Users can easily print the page contents by clicking on the Print button in the toolbar.

**Multithreading**: Multithreading is used in this application. Every tab is opened in a new thread with a background worker delivering the contents after receiving the http response. Users are able to request multiple pages at once.

**4. User Guide**

In this user guide, I will try to set you up with the user interface for Meteora in minimum number of steps as possible. First, let’s have a look at the basic layout of the browser.



***Fullscreen:***

Go fullscreen.

***Print:***

Print the contents.

***Address bar:***

Type the URL here.

***Home page:***

Clicking on this button will take you to the homepage.

***Refresh:***

Refresh the contents.

***Menu Bar:***

The menus.

***Browse:***

Click to browse.

***Add to Favorites:***

Add this URL to favorites.

***Html contents:***

The contents of the web page or an error message will appear here..

***Tabs:***

Select the different tabs from here..

***Server Information:***

Displays the server info.

***Status Code:***

Displays the status code of the request.

***Render:***

Render the entered URL using webbrowser control.

A brief description of the buttons are given below:

**Refresh Button:** Refresh the contents on current tab by resending the http request.

**Fullscreen Button:** The full screen button will display the browser in full screen.

**Home page Button:** Takes you to the home page on the current tab.

**Address Bar:** Type the URL you want to visit here. Pressing the enter key after typing the URL will also start browse.

**Browse Button:** The user can click on the browse button after entering the URL in the address bar.

**Print Button:** This will open the print dialog and prints the entire contents of the page.

**Add to Favorites Button:** Click on this button to add this page to your favorites. This will open the Add to favorites window, where you’ll have to enter the name for the link.

**Tabs:** You can select the different open tabs from here.

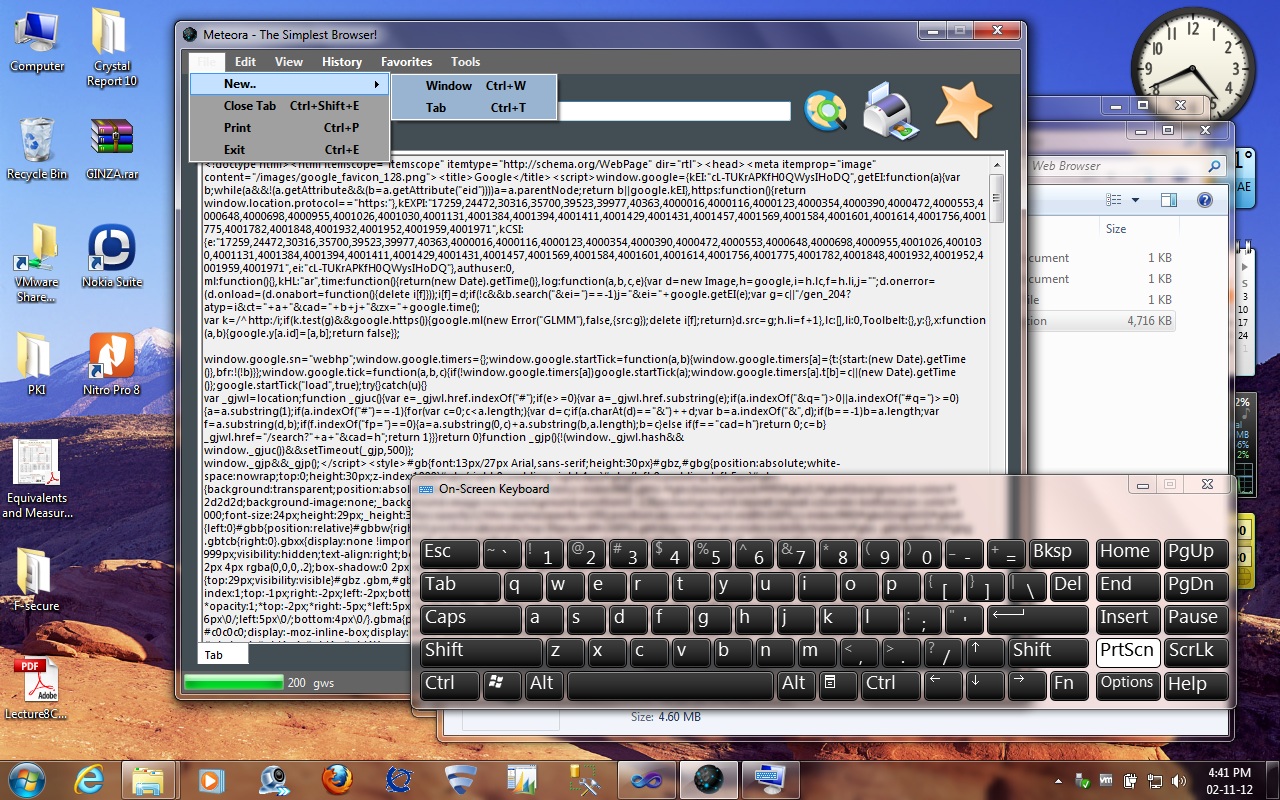
**HTML Contents:** The html contents or the error message will be displayed here.

**Status Code:** The status code of the http request will be displayed here.

**Server Info:** The server information for the current http request will be displayed here.

**Render:** This will render the current tab using the Web browser control.

Now coming to the different menus of the browser. Let us begin with the file menu.



From this menu, you can:

1. Open a new Window.

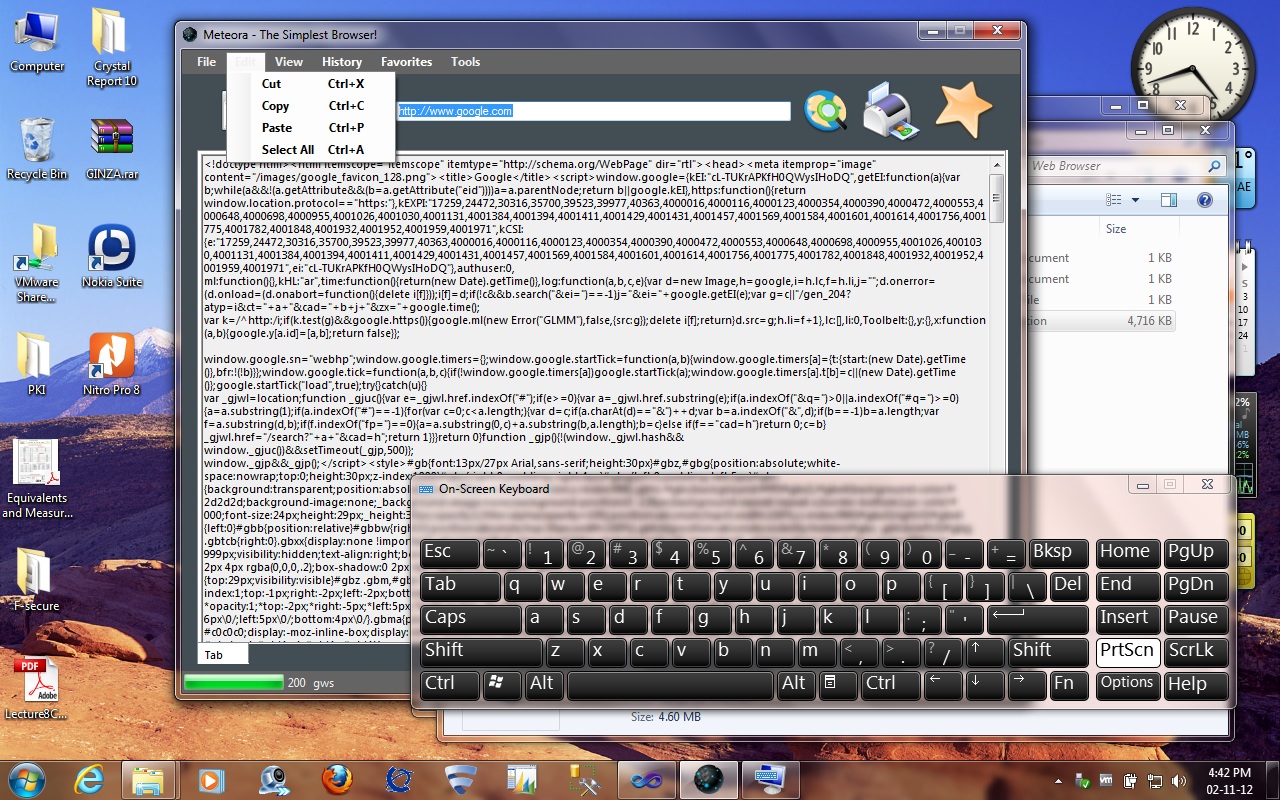
2. Open a new tab.

3. Close the current tab.

4. Print the current tab contents.

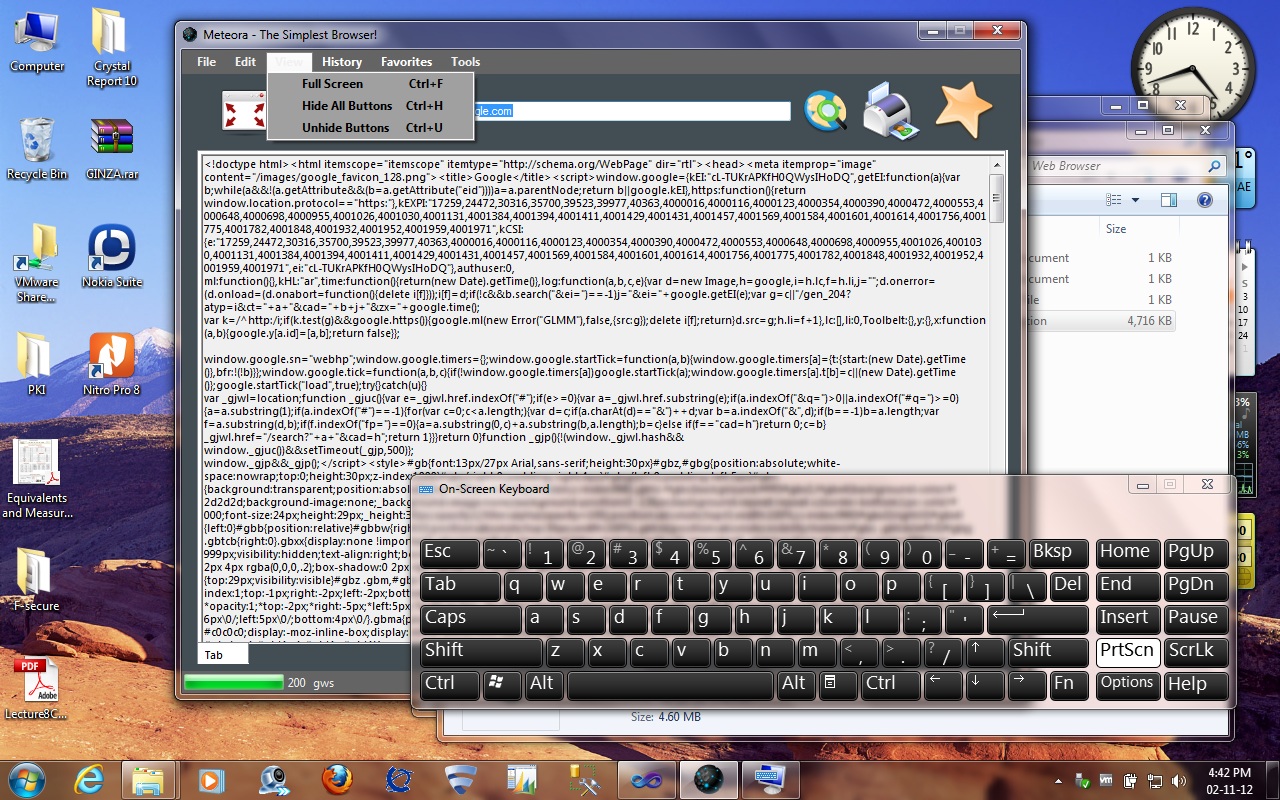
5. Exit Meteora.

Now, let move on to the Edit menu. The edit menu contains all the standard options such as:

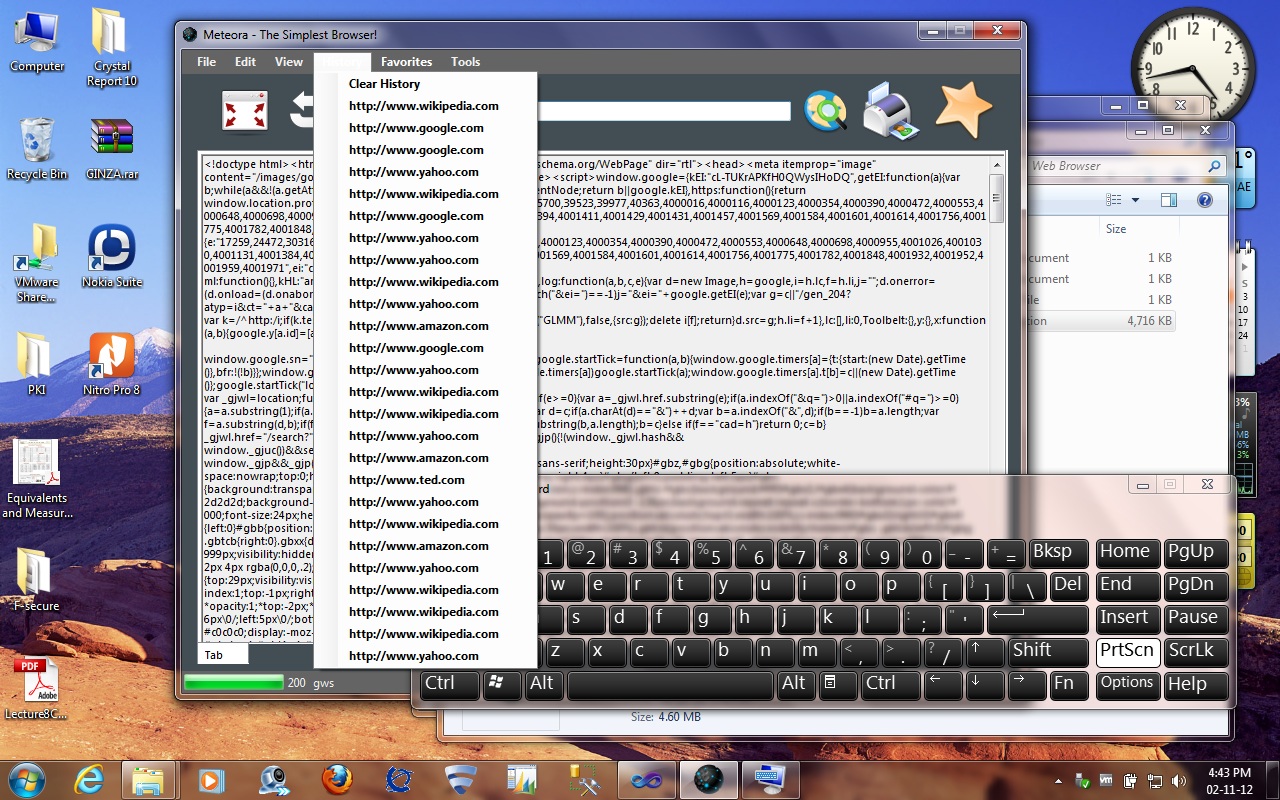


1. **Cut:** Cuts the selected text.
2. **Copy:** Copies the selected text.
3. **Paste:** Pastes the selected text.
4. **Select All:** Select All the text in the current window.

The View Menu contains three options, which are:

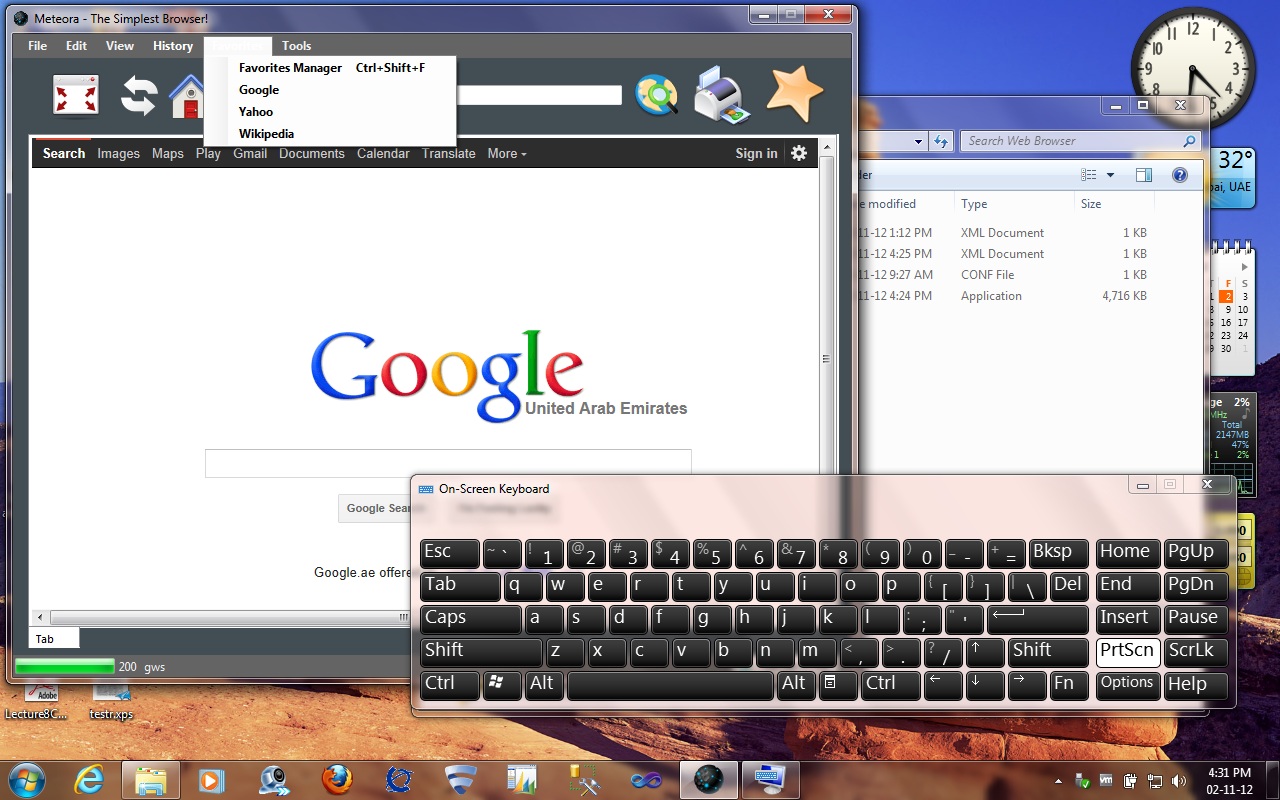


1. **Full Screen:** Puts your browser into full screen.
2. **Hide All Buttons:** Hides all the buttons in the browser.
3. **Unhide Buttons:** Undo the actions of previous option.



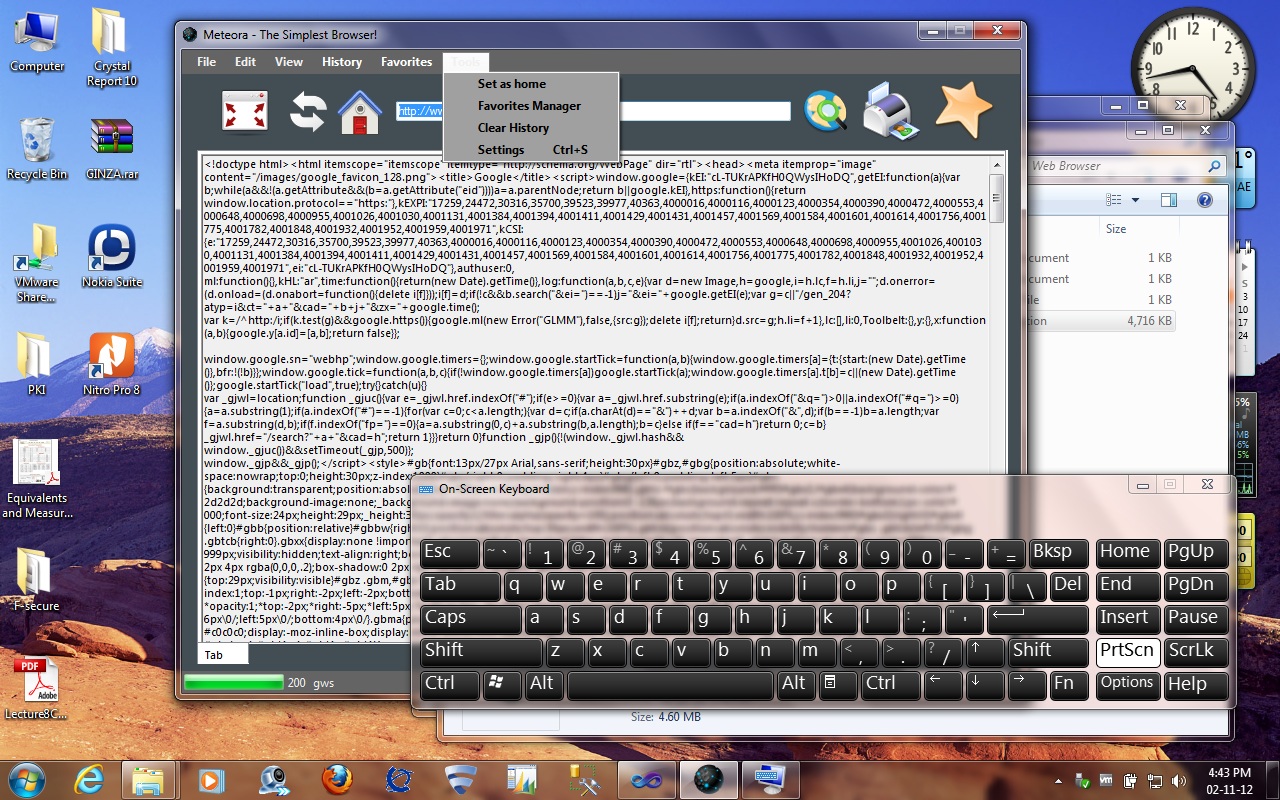
The history menu contains all the previously visited URLs. These links are dynamically loaded into the browser on page load and whenever there is a change.

Clear history: This will remove all the history from the browser.



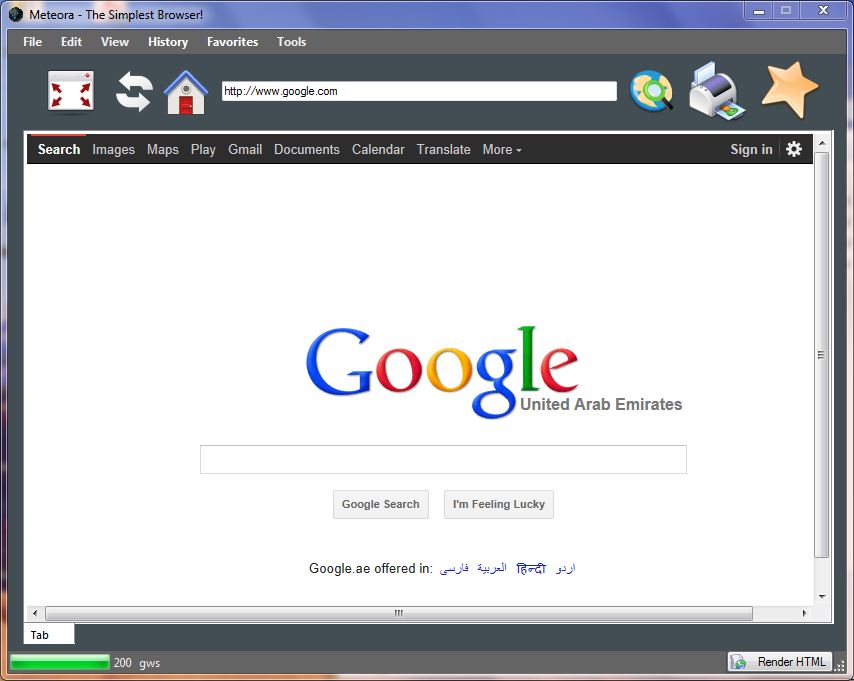
The favorites menu provides easy access to all the favorites added to the browser. This menu also contains an option to open the favorites manager.

The favorites manager is used to add, edit and remove favorites.

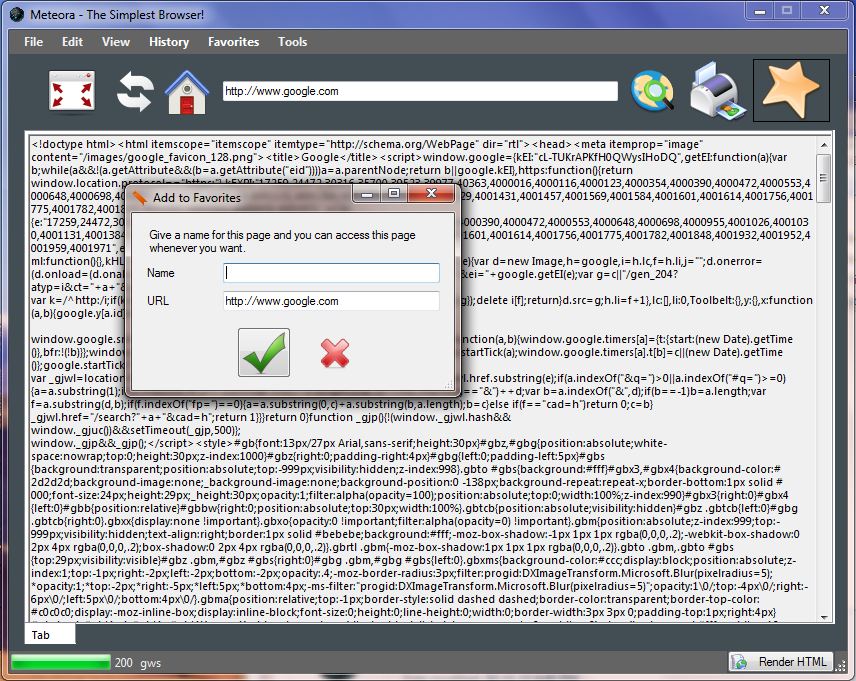
The tools menu contains the following options:

1. **Set as Home:** This will set the current page as the homepage.
2. **Favorites Manager:** This will open the favorites manager.
3. **Clear History:** Clears all the history.
4. **Settings:** Opens the settings window.

**Add Favorites**

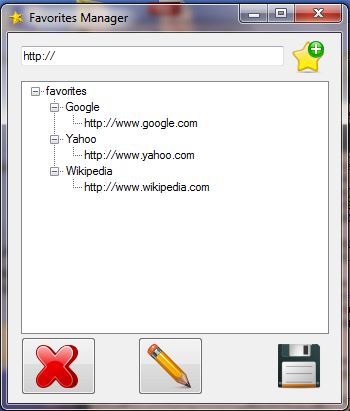


When you click on the Add favorite button, the Add to Favorites window will open. The URL of the page will be automatically loaded into the window. You will just have to enter the name for the favorite. After you finish, just click on the tick mark to confirm addition.



**Favorites Manager**

The Favorites Manager is the place where you have complete control over each and everything in your favorites. Here you have the option to change the name or even edit the URL for a link. A screenshot of the favorites manager is shown below.

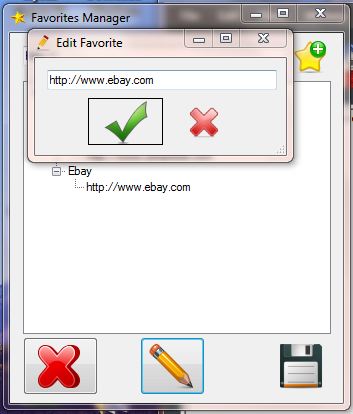


***Remove Favorite***

***Edit Favorite***

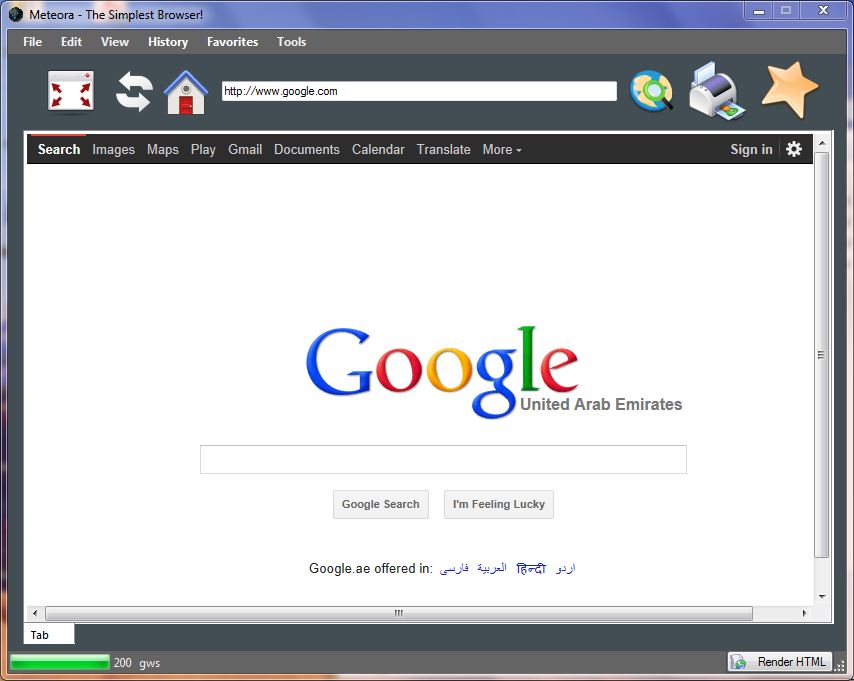
***Add Favorite***

***Save & Exit***

In the favorites manager you can Add, Edit or Remove any favorites. Click on any URL or name you want to edit and then click the Pencil icon to open the edit favorites window. The edit favorite window is displayed below.

After making the required modifications, click on Save and Exit.

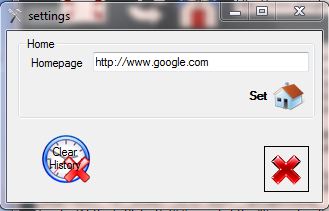
**Print Dialog**



Clicking on the print button will open the print dialog. This is the standard print dialog you will get with any other application. Clicking on print will print all the contents of the selected tab. You have the option to give the number of copies, the required pages,.. in this dialog.

**Settings**

The settings window is fairly basic and contains very limited functionalities. It is just a placeholder for future upgrades. A screenshot of the settings window is given below.



***Set Home:*** Click on this button will set the entered URL as the homepage.

***Close:*** Close the settings window.

***Clear History:*** This will clear all the history.

There are basically two options in the settings window, they are:

**Set Home:** Set the URL in the textbox as the homepage. The home page will be loaded on new tabs when the application starts.

*You may have to restart the browser for this to take effect.*

**Clear History:** Clears all the history from the browser.

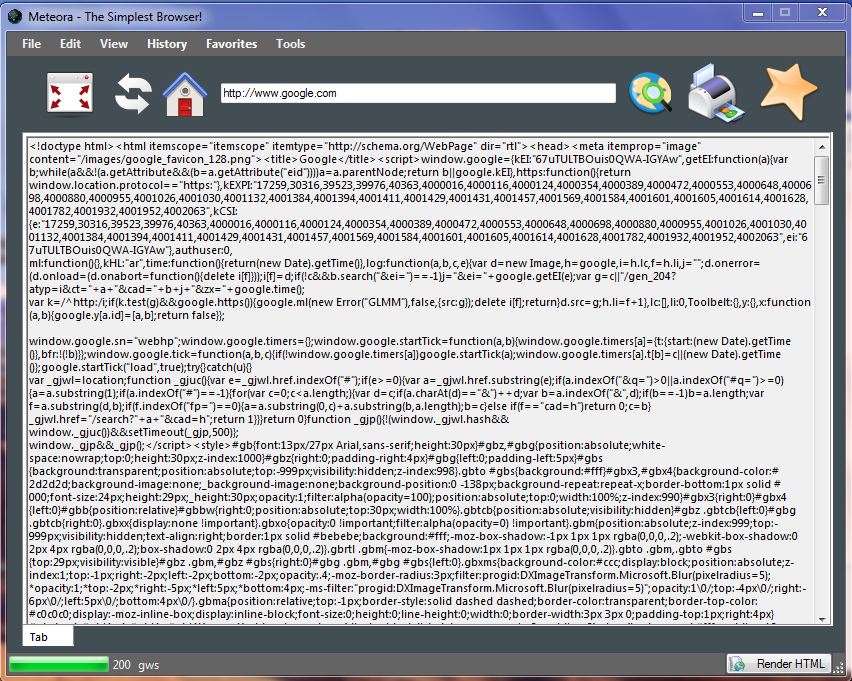
**The various shortcut keys used in my application are given below:**

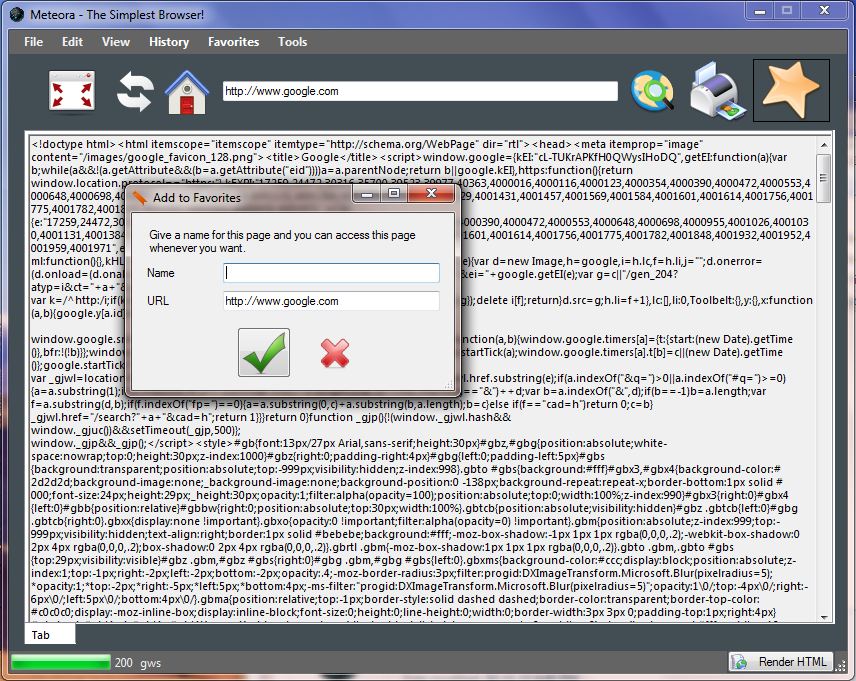
|  |  |  |
| --- | --- | --- |
| ***Short keys*** | ***Name*** | ***Description*** |
| **Ctrl+T** | *New Tab* | Opens a new tab. |
| **Ctrl+W** | *New Window* | Opens a new window. |
| **Ctrl+P** | *Print* | Prints the contents of current tab. |
| **Ctrl+Shift+E** | *Close Tab* | Closes the current tab. |
| **Ctrl+E** | *Exit* | Closes Meteora. |
| **Ctrl+F** | *Full screen* | Makes the application fullscreen. |
| **Ctrl+H** | *Hide all buttons* | Hides all the buttons. |
| **Ctrl+U** | *Unhide buttons* | Unhides the hidden button. |
| **Ctrl+Shift+H** | *Clear History* | Clears all the history. |
| **Ctrl+Shift+F** | *Favorites Manager* | Opens the favorites manager. |
| **Ctrl+S** | *Settings* | Opens the browser settings. |

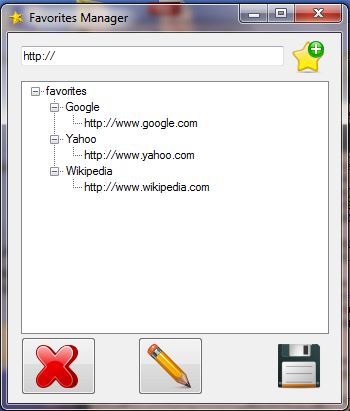
**5. Developer Guide**

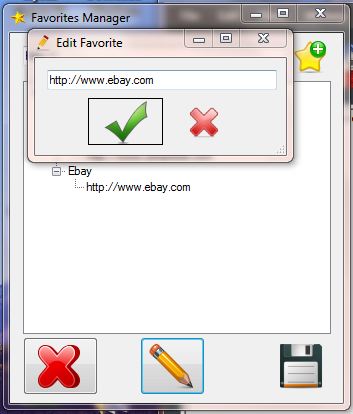
In this guide, I will explain the basic structure and working of my web browser from a developer’s point of view and the major codes and functions used.

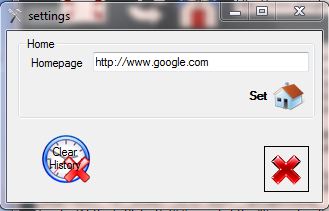
This is a very simple and basic application with only 5 windows forms involved. The forms are:

1. ***Browser.cs:*** This file contains the code and design for the main browser form. This form is the only one called from the Program.cs file, ie. It is the only form loaded on application launch. A screenshot of browser form is shown below.



1. ***Addfav.cs:*** This form is used to add new favorites to the application. It is a very basic form with name and address textboxes. The address will be automatically passed to this form from the calling function. The user has to just enter the name for this link. A screenshot of this form is given below.
2. ***Favorites.cs:*** This is the Favorites Manager Form. This form is used to add, update and remove favorites. Users can edit either the name or the URL of a favorite individually. The favorites.xml is loaded directly into a Treeview control in this form and when the user clicks ‘Save’, the treeview control as such is copied to the XML file, thereby, updating the favorites.xml file. A screenshot of this form is given below.



1. ***Editfav.cs:*** This is used to edit the values from the favorites manager. The text in this window is passed by the calling function. It will be the text selected by the user.
2. ***Settings.cs:*** This form contains few settings for the application such as set as home and clear history.

Now, Ill give a brief explanation on the working of this application.

When the application is first launched, the form browser.cs is opened. On form load, a new tab is created with the contents of the homepage. The new tab creation process goes through 2 functions and one background worker. A new tab is created on a new thread. The new\_tab() function creates a tab page and adds the required controls such as the textbox and additional information to it. It then calls the background worker. The background worker calls the httpfunction().

The httpfunction() function, creates a temporary textbox and uses the httprequest and httpresponse functions to get the html content from the entered URL. The URL along with the tab number is passed to the httpfunction() by the background worker. The html contents are then loaded to the rich text box control in the specified tab.

I have XML files to store the favorites and history data for the application. There are two xml files used in this application, Favorites.xml and History.xml. The XML files are actively used throughout the application. The favorites menu will be updated as soon as a new item is added. Modifications to the favorites can be done using the favorites manager. The favorites manager loads the entire xml into a Treeview control. The user can add, edit and remove favorites easily using the favorites manager. When the user clicks on save, the entire Treeview nodes will be completely saved to the xml file, thereby updating the favorites. After calling the favorites manager, the updatefavorites() function is called to update the favorites menu. The structure of favorites.xml is as shown below:

<?xml version="1.0" encoding="utf-8" ?>

<favorites>

<Google>http://www.google.com</Google>

<Yahoo>http://www.yahoo.com</Yahoo>

</favorites>

The history menu will automatically update its content while the application is running. The history.xml will be completely cleared on clicking clear history from the tools menu or from the settings. The structure of history.xml is shown below:

<history>

<link>http://www.linkedin.com</link>

</history>

A .conf file is used to store the homepage address outside of the application. The config file will be loaded from program.cs and the value will be passed to the browser object. The config file will be updated by two actions: Clicking on the Set as home button in tools menu or by changing homepage in the settings. The .conf file can be configured to save more settings as the application grows.

Some of the major functions used in this application are,

**new\_tab(string urlvalue) Function:** This function is first called while creating a new tab. It creates a new tab with a rich text box control. The contents are later loaded into this control.

public void new\_tab(string urlvalue)

{

Label addresslbl = new Label();

string addr = "http://";

int t = tabno;

tabno++;

TabPage newtab = new TabPage("Tab");

RichTextBox contentstext= new RichTextBox();

contentstext.Width = tabs.Width-5;

contentstext.Height = tabs.Height-20;

contentstext.Multiline = true;

contentstext.ReadOnly = true;

contentstext.Text = "";

//Checking if this is the first tab, then display homepage

if (urlvalue == "http://")

{

addr = hp;

}

else

{

addr = urlvalue;

}

//Initializing the background workers

var bw = new BackgroundWorker();

bw.WorkerReportsProgress = true;

bw.WorkerSupportsCancellation = true;

bw.DoWork += delegate

{

var bwth = new Thread(() => reportprog(bw));

bwth.Start();

httpfunction(addr,t);

bwth.Join();

bw.CancelAsync();

};

bw.ProgressChanged += new ProgressChangedEventHandler(bw\_ProgressChanged);

var bw2 = new BackgroundWorker();

bw2.WorkerReportsProgress = true;

bw2.WorkerSupportsCancellation = true;

bw2.DoWork += delegate

{

var bw2th = new Thread(() => reportprog(bw2));

bw2th.Start();

httpfunction(addr, t);

bw2th.Join();

bw2.CancelAsync();

};

bw2.ProgressChanged += new ProgressChangedEventHandler(bw\_ProgressChanged);

if (bw.IsBusy)

bw2.RunWorkerAsync();

else

bw.RunWorkerAsync();

newtab.Controls.Add(contentstext);

Label tnolb = new Label();

tnolb.Text = t.ToString();

this.Invoke(new MethodInvoker(delegate {addresslbl.Text = addr;}));

this.Invoke(new MethodInvoker(delegate { addresstb.Text = addr; }));

this.Invoke(new MethodInvoker(delegate {newtab.Controls.Add(addresslbl);}));

this.Invoke(new MethodInvoker(delegate { newtab.Controls.Add(tnolb); }));

this.Invoke(new MethodInvoker(delegate {tabs.TabPages.Add(newtab);}));

this.Invoke(new MethodInvoker(delegate { tabs.SelectTab(tabs.TabCount - 1); }));

}

This function creates a new tabpage, a textbox and two labels, to store the address and the tab number to each tab. It will check if the address box is empty or not and If it is, will open the homepage. Then add all the controls to the tab. It will then call a background worker. It will also add the new tab to the Tabcontrol.

**Httpfunction(string addr) Function:** Function to do the http request and response.

public void httpfunction(string addr, int tbn)

{

RichTextBox contents = new RichTextBox();

contents.Text = "";

try

{

//Initializing the url

Uri url = new Uri(addr);

//HTTP Initialization

HttpWebRequest WebReq = (HttpWebRequest)WebRequest.Create(url);

WebReq.Method = "GET";

// WebReq.Timeout = 6500;

HttpWebResponse WebResp = (HttpWebResponse)WebReq.GetResponse();

//Gets information about the response

Stream postdata = WebResp.GetResponseStream();

int sc;

sc = (int)WebResp.StatusCode;

//Writes the status code & server to the toolstrip

status.Text = sc.ToString();

server.Text = WebResp.Server;

//Now, we read the response (the string), and output it.

Stream Answer = WebResp.GetResponseStream();

StreamReader \_Answer = new StreamReader(Answer);

contents.Text = \_Answer.ReadToEnd();

for (int i = 0; i < tabs.TabCount; i++)

{

if (tabs.TabPages[i].Controls[2].Text == tbn.ToString())

{

this.Invoke(new MethodInvoker(delegate { tabs.TabPages[i].Controls[0].Text = contents.Text; }));

}

}

}

catch (Exception e)

{

for (int i = 0; i < tabs.TabCount; i++)

{

if (tabs.TabPages[i].Controls[2].Text == tbn.ToString())

{

this.Invoke(new MethodInvoker(delegate { tabs.TabPages[i].Controls[0].Text = e.Message; }));

}

}

}

}

This function first creates a temporary textbox and then converts the passed string to a URL. This URL is then used to call the httprequest function. The http response is then put to a stream. This stream will be then copied to the text box control in the specified tab. This function also gives the errors with the error codes through exception handling.

**Refreshtab( ) Function:** This function is used to refresh a tab when the refresh button is pressed.

public void refreshtab() {

try

{

int tno;

tno = tabs.SelectedIndex + 1;

string address;

address = addresstb.Text;

tabs.TabPages[tabs.SelectedIndex].Controls[0].Visible = true;

var rth = new Thread(() => httpfunction(address,tno));

rth.Start();

backgroundWorker1.RunWorkerAsync();

}catch (Exception exc)

{

tabs.TabPages[tabs.SelectedIndex].Controls[0].Text = exc.Message;

}

}

Its function is same as new\_tab() function but here we don’t create a new tab, the contents of the selected tab is updated.

**Updatefavorites() & updatehistory() Functions:** These are used to load the contents from the XML files to the menus.

public void updatehistory()

{

var clrhistory = historymenu.DropDownItems[0];

this.Invoke(new MethodInvoker(delegate { historymenu.DropDownItems.Clear(); }));

this.Invoke(new MethodInvoker(delegate { historymenu.DropDownItems.Add(clrhistory); }));

XmlDocument xhistoryload = new XmlDocument();

xhistoryload.Load("history.xml");

XmlNodeList hnodelist = xhistoryload.SelectNodes("//history/link");

foreach (XmlNode node in hnodelist)

{

ToolStripMenuItem newfav = new ToolStripMenuItem();

newfav.Text = node.InnerText;

this.Invoke(new MethodInvoker(delegate { historymenu.DropDownItems.Add(newfav); }));

}

}

The updatehistory() function is used to load the history items to the history menu automatically when a change occurs in history.xml. It first opens history.xml, and then adds all the links to the history menu. The updatefavorites() function provides the same functionality for the favorites menu.

**Addhistory(string url) Function:** This function adds a new URL to the history.

public void addhistory(string url)

{

XmlDocument myXml = new XmlDocument();

XmlElement el = myXml.CreateElement("link");

el.InnerText = url;

if (!File.Exists("history.xml"))

{

XmlElement root = myXml.CreateElement("history");

root.AppendChild(el);

myXml.AppendChild(root);

}

else

{

myXml.Load("history.xml");

myXml.DocumentElement.AppendChild(el);

}

myXml.Save("history.xml");

updatehistory();

}

private void addfavb\_Click(object sender, EventArgs e)

{

addfav adddiag = new addfav(addresstb.Text.ToString(), "");

DialogResult favresult = adddiag.ShowDialog();

if (DialogResult == 0)

clearupdatefavmenu();

}

The addhistory() function add the passed string to the history.xml file. It also calls the updatehistory() function, so that the history menu is updated.

**Clearhistory() Function:** This function clears the history items from the XML.

public void clearhistory() {

XmlDocument myXml = new XmlDocument();

if (!File.Exists("history.xml"))

{

XmlElement root = myXml.CreateElement("history");

myXml.AppendChild(root);

}

else

{

myXml.Load("history.xml");

// myXml.SelectSingleNode("history");

myXml.RemoveAll();

XmlElement root = myXml.CreateElement("history");

myXml.AppendChild(root);

}

myXml.Save("history.xml");

//Reload all the history items on menu

var clrhistory = historymenu.DropDownItems[0];

historymenu.DropDownItems.Clear();

historymenu.DropDownItems.Add(clrhistory);

updatehistory();

}

The clear history function is used to clear all links from the history.xml. It opens history.xml, clears all items and then creates just the root element. It again calls the updatehistory() function.

**Fullscreen() Function**: This function makes the application goes full screen.

void Fullscreen()

{

try

{

if (fullscreen == false)

{

this.restore.location = this.Location;

this.restore.width = this.Width;

this.restore.height = this.Height;

this.restore.tabw = this.tabs.Width;

this.restore.tabh = this.tabs.Height;

this.restore.rendbp = this.renderb.Location;

this.restore.tbw =this.tabs.TabPages[tabs.SelectedIndex].Controls[0].Width;

this.restore.tbh = this.tabs.TabPages[tabs.SelectedIndex].Controls[0].Height;

this.TopMost = true;

this.FormBorderStyle = FormBorderStyle.None;

this.Width = Screen.PrimaryScreen.Bounds.Width;

this.Height = Screen.PrimaryScreen.Bounds.Height;

this.Location = new Point(0,0);

this.tabs.Width = this.Width - 40;

this.tabs.Height = this.Height -130;

this.tabs.TabPages[tabs.SelectedIndex].Controls[0].Width = this.tabs.Width - 10;

this.tabs.TabPages[tabs.SelectedIndex].Controls[0].Height= this.tabs.Height - 30;

this.renderb.Location = new Point(this.Width - 125, this.Height - 23);

fullscreen = true;

}

else

{

this.TopMost = false;

this.Location = this.restore.location;

this.Width = this.restore.width;

this.Height = this.restore.height-40;

this.tabs.Width = this.restore.tabw;

this.tabs.Height = this.restore.tabh;

this.renderb.Location = this.restore.rendbp;

this.tabs.TabPages[tabs.SelectedIndex].Controls[0].Width = this.restore.tbw ;

this.tabs.TabPages[tabs.SelectedIndex].Controls[0].Height = this.restore.tbh;

this.WindowState = FormWindowState.Normal;

this.FormBorderStyle = FormBorderStyle.Sizable;

fullscreen = false;

}

}catch(Exception){}

}

This function is used to make the form fullscreen. It first stores all the current values of the form to a structure and then maximizes the form. It makes use of a Boolean variable fullscreen to detect if the application is now in fullscreen or not.

**Render Function:** It will render the webpage to the current tab.

private void renderb\_Click(object sender, EventArgs e)

{

try

{

//Get the current selected tab index

int tabno = tabs.SelectedIndex;

//create a WebBrowser object

WebBrowser browser = new WebBrowser();

browser.Navigate(addresstb.Text.ToString());

browser.Dock = DockStyle.Fill;

tabs.TabPages[tabno].Controls[0].Visible = false;

tabs.TabPages[tabno].Controls[1].Visible = false;

tabs.TabPages[tabno].Controls.Add(browser);

}

catch (Exception){}

}

The render button adds a new web browser control into the existing tab. This control will render the HTML and display the webpage the way it was meant to be shown.

**Closetab Function**: This menu item is used to close an open tab.

private void closeTabToolStripMenuItem\_Click(object sender, EventArgs e)

{

TabPage tb = tabs.SelectedTab;

tabs.TabPages.Remove(tb);

if (tabs.TabCount == 0)

{

if (MessageBox.Show(this, " There are no other open tabs. Close Meteora?", "Close Meteora", MessageBoxButtons.YesNo) == DialogResult.Yes)

{

this.Close();

}

else

{

new\_tab(hp);

addresstb.Text = hp;

}

}

}

This is a pretty basic function used to close a tab. It also checks it the current tab is the last one, if so, it will present a dialog box asking whether to close the application or open a new tab.

**Populatetree() Function:** You will find this function in the favorites.cs form. This function is used to load the contents of favorites.xml to a treeview control. It first opens the xml file and creates the first treeview node.

public void populatetree()

{

//Create a DOM Document and load the XML data into it.

XmlDocument dom = new XmlDocument();

dom.Load("favorites.xml");

//Initialize the TreeView control.

treeView1.Nodes.Clear();

treeView1.Nodes.Add(new TreeNode(dom.DocumentElement.Name));

TreeNode tNode = new TreeNode();

tNode = treeView1.Nodes[0];

//Populate the TreeView with the DOM nodes.

AddNode(dom.DocumentElement, tNode);

treeView1.ExpandAll();

}

**AddNode(XmlNode inXmlNode, treeNode inTreeNode) Function:** This function is also found in the favorites.cs form. It adds all the XML nodes to the treeview control. It reiterates until a leaf is reached. It will call itself each time a child node is detected.

private void AddNode(XmlNode inXmlNode, TreeNode inTreeNode)

{

XmlNode xNode;

TreeNode tNode;

XmlNodeList nodeList;

int i;

// Loop through the XML nodes until the leaf is reached.

// Add the nodes to the TreeView during the looping process.

if (inXmlNode.HasChildNodes)

{

nodeList = inXmlNode.ChildNodes;

for(i = 0; i<=nodeList.Count - 1; i++)

{

xNode = inXmlNode.ChildNodes[i];

TreeNode tn = new TreeNode(xNode.Name);

inTreeNode.Nodes.Add(tn);

tNode = inTreeNode.Nodes[i];

AddNode(xNode, tNode);

}

}

else

{

// Here you need to pull the data from the XmlNode based on the

// type of node, whether attribute values are required, and so forth.

inTreeNode.Text = (inXmlNode.OuterXml).Trim();

}

}

**ExporttoXml (TreeView tv, string filename) Function:** This function is used to export the treeview to the XML file and update the XML file.

public void exportToXml(TreeView tv, string filename)

{

xr = new XmlTextWriter(filename, System.Text.Encoding.UTF8);

xr.WriteStartDocument();

//Write our root node

xr.WriteStartElement(treeView1.Nodes[0].Text);

foreach (TreeNode node in tv.Nodes)

{

saveNode(node.Nodes);

}

//Close the root node

xr.WriteEndElement();

xr.Close();

}

**addFavorite(String url, string name) Function:** This function is found in Addfav.cs. This function is used to add a new favorite to the xml file. It takes two values, the URL and the name for the favorite and then writes it to the favorites.xml file.

private void addFavorite(String url, string name)

{

XmlDocument myXml = new XmlDocument();

XmlElement el = myXml.CreateElement(name);

el.InnerText = url;

if (!File.Exists("favorites.xml"))

{

XmlElement root = myXml.CreateElement("favorites");

myXml.AppendChild(root);

root.AppendChild(el);

}

else

{

myXml.Load("favorites.xml");

myXml.DocumentElement.AppendChild(el);

}

myXml.Save("favorites.xml");

}

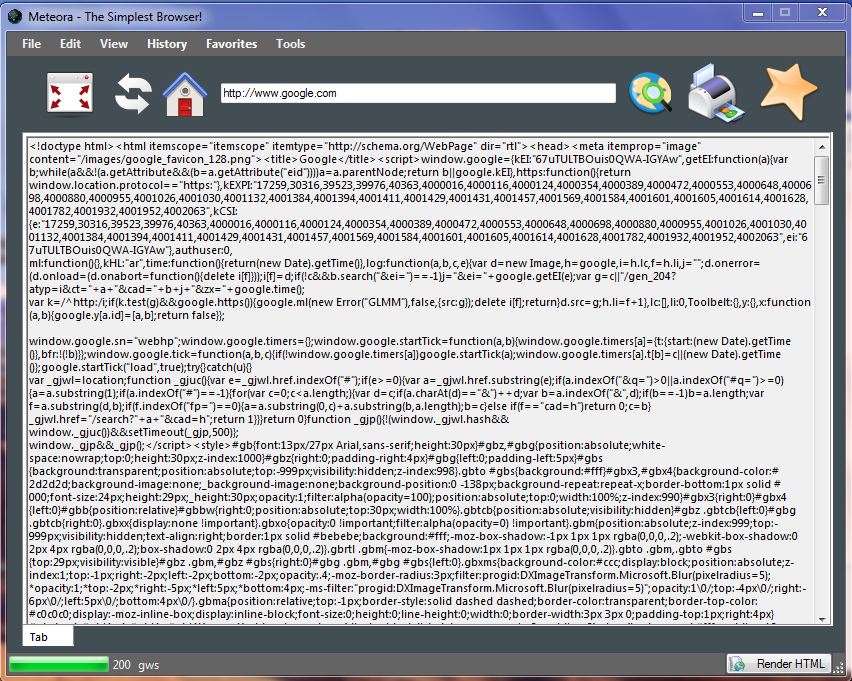
**Multithreading**

In this application, each tab has its own thread. The updatefavorites() and updatehistory() functions also run on different threads. In addition to these threads there are three background workers aiding the application to run smoothly. Two of them are used in the new\_tab() function during the creation of a new tab. They run side by side and runs the httpfunction() function in background and then fills html contents. The other background worker is used in a similar scenario for the refreshtab() function.

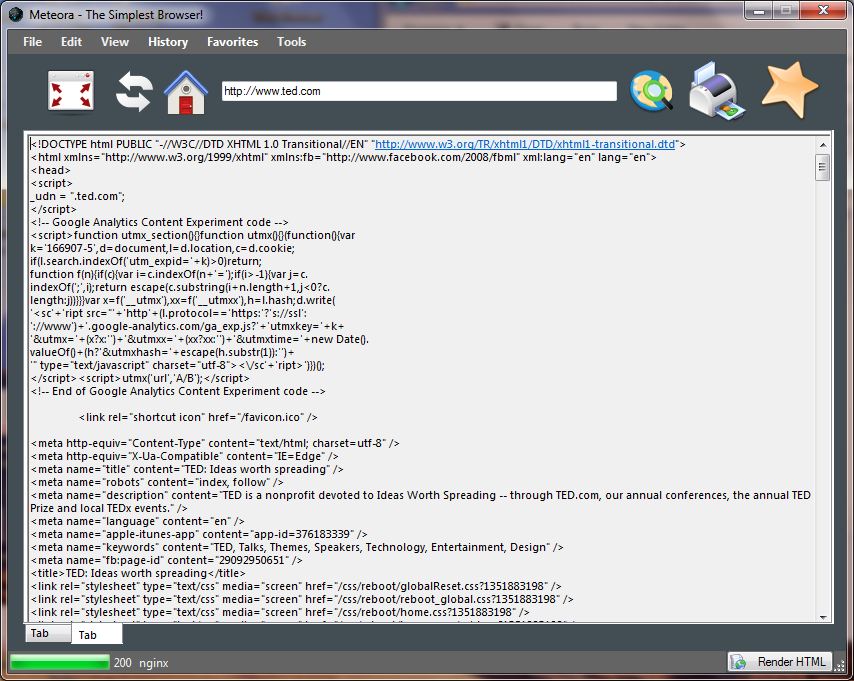
Further work needs to be done to streamline and improve the performance of the threads and background workers.

**6. Testing**

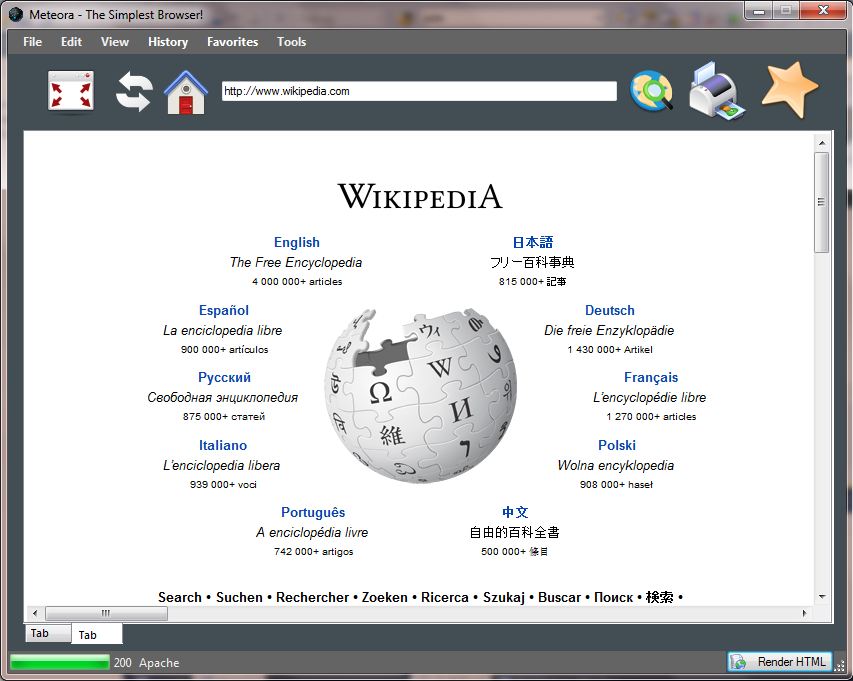
When the application loads, you will be presented with the following screen with the homepage.



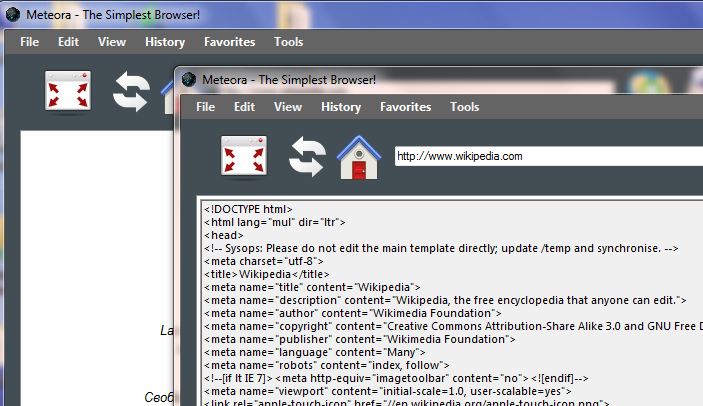
Now let us begin testing each function, one by one. Typing an address in the address bar and pressing enter will open a new tab and take you to the page.



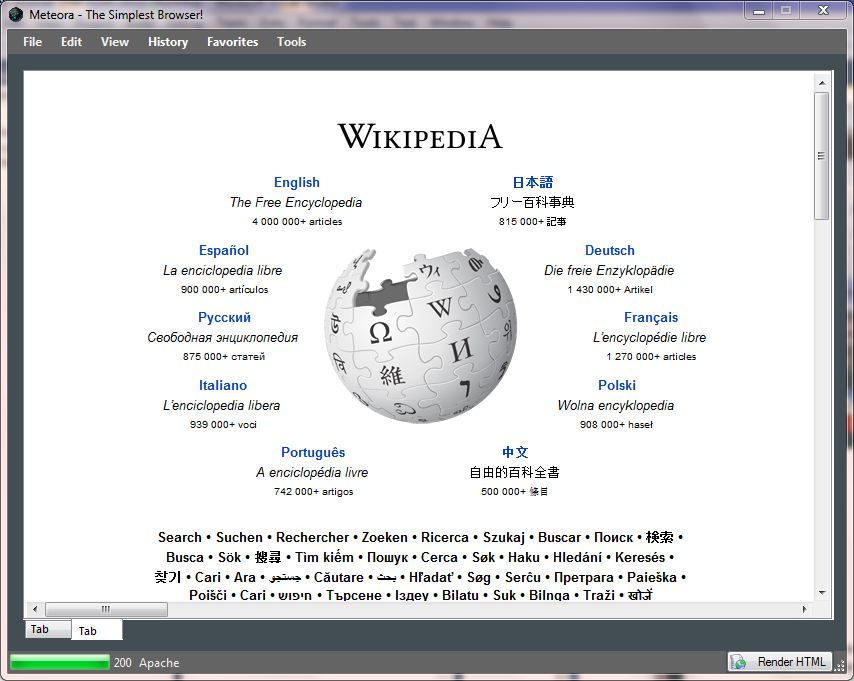
When you click the render button while on a page, will render the page using the C# web browser class.



Clicking on New Window from the file menu, will open another instance of the Meteora.

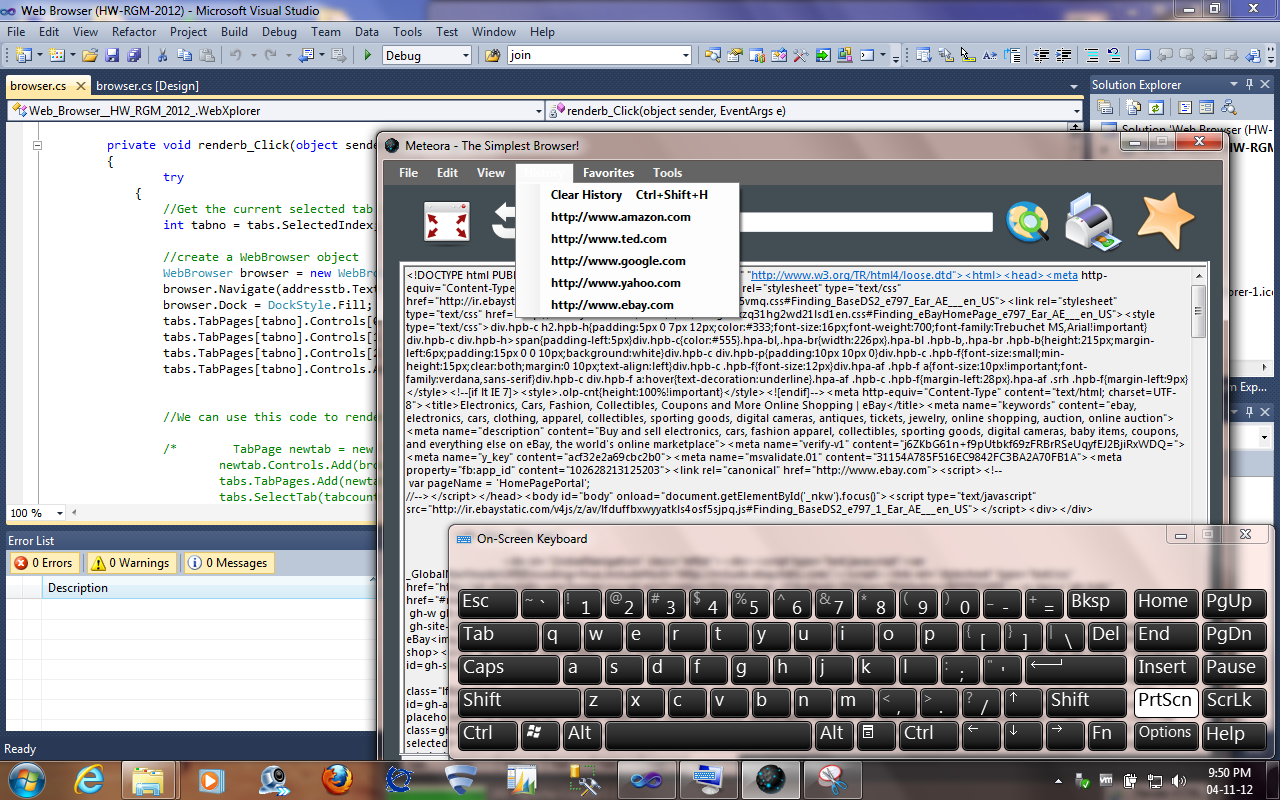


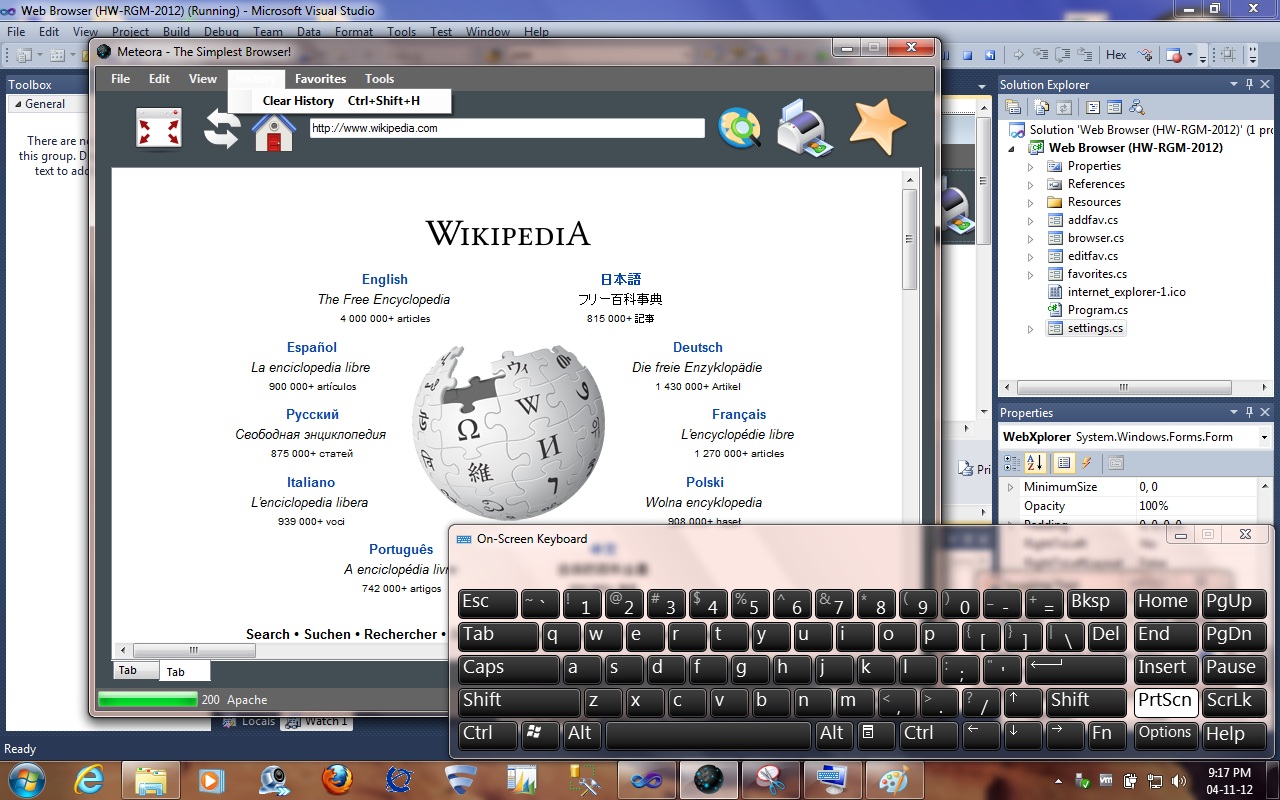
When you click Hide all buttons under the View menu, all the buttons will be hidden as shown.



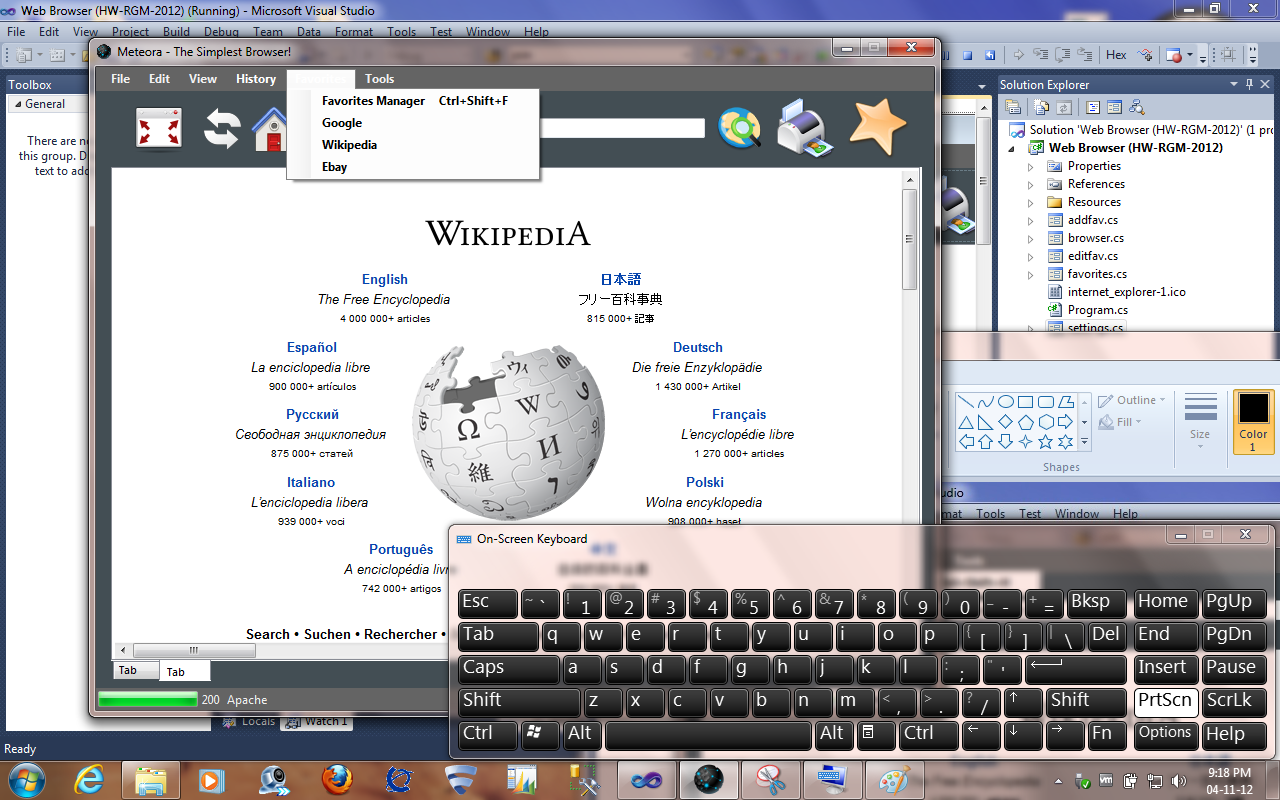
Clicking on unhide buttons will restore all the buttons back.

When you click on history menu, all the history will be automatically loaded into it as shown.



Now, if you click on clear History, it will clear all the items from the history.

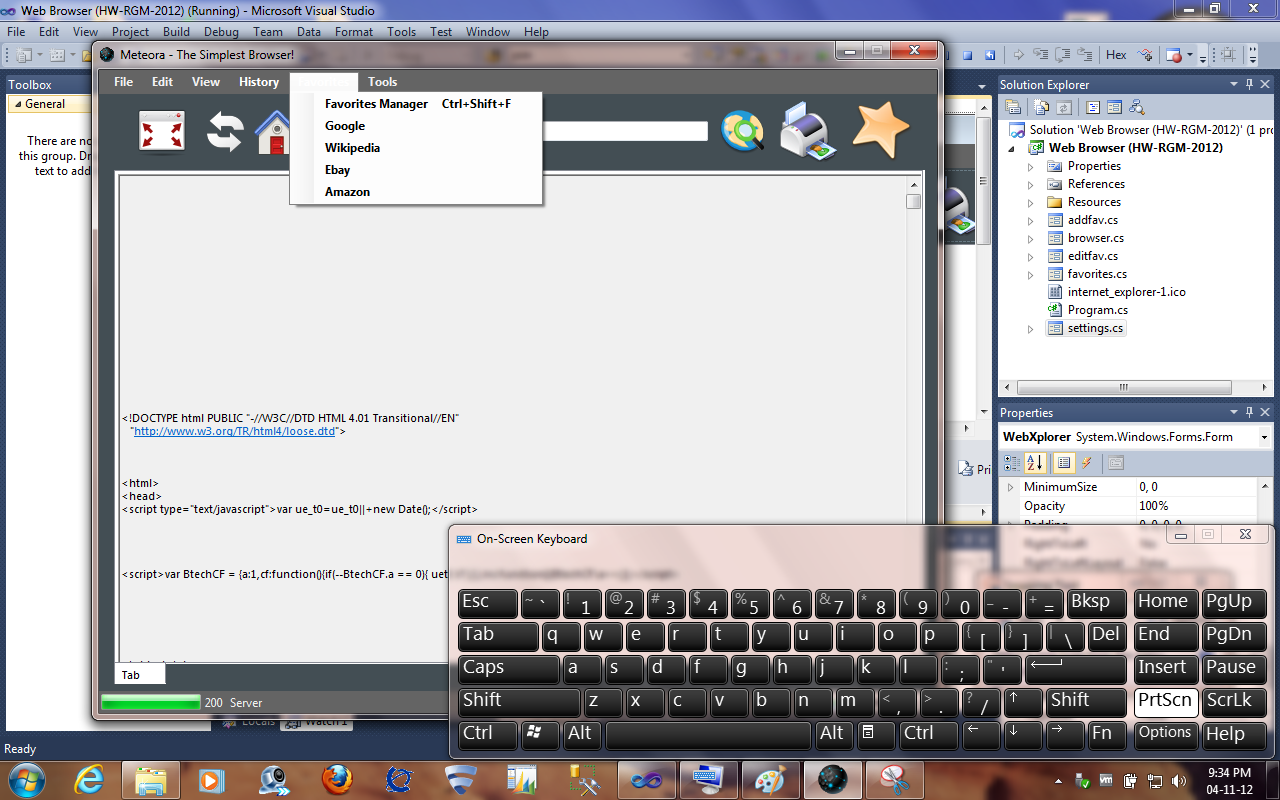
Now let us test, the Add to Favorites and the Favorites Manager. Before adding the favorites menu looks like this:



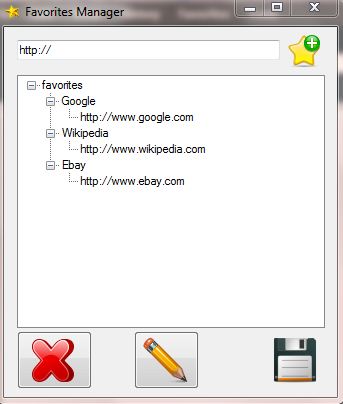
Now click on Add to Favorites icon and add a link. *Please note that adding a space in the favorite name can cause errors.*



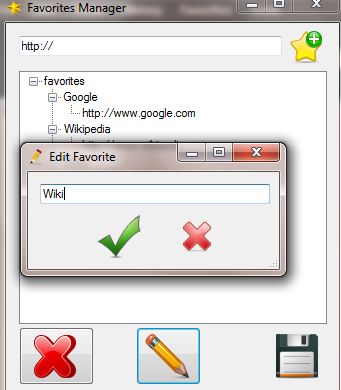
After adding the favorites menu will look like this.



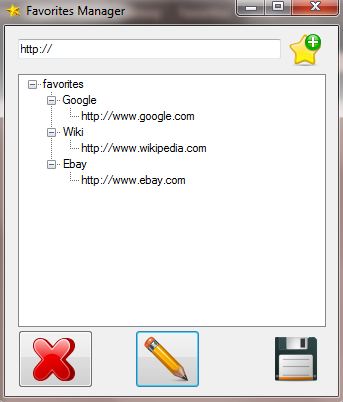
Now we will open the Favorites Manager and make a modification an existing favorite.



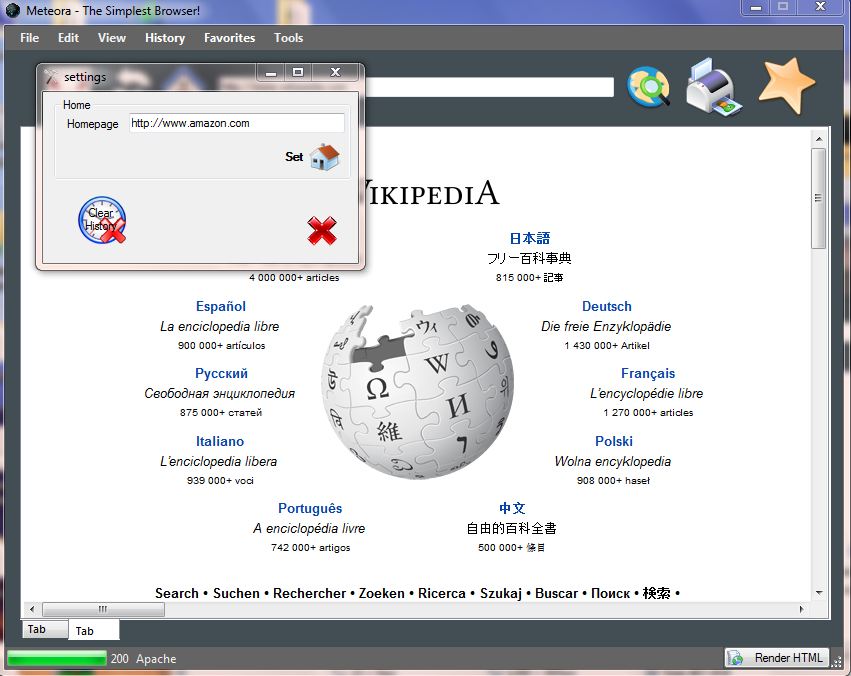
Click on Amazon and click on the pencil icon. This will open the Edit favorite window as shown.



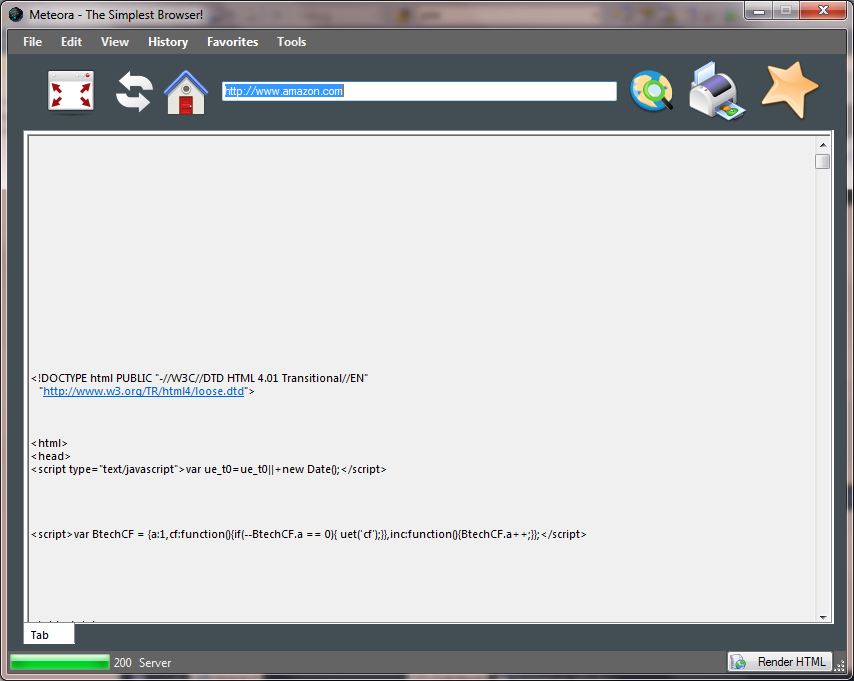
The favorites manager after clicking ok will look like this.

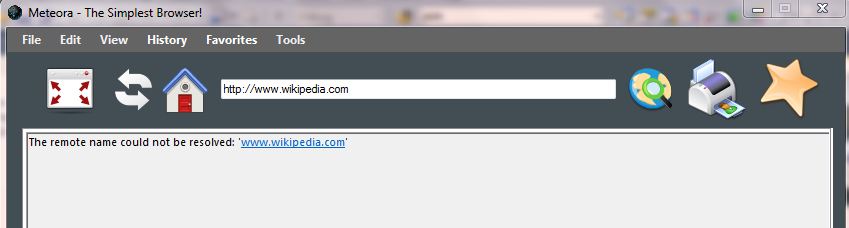


Now, let us open the settings window and set a new homepage.

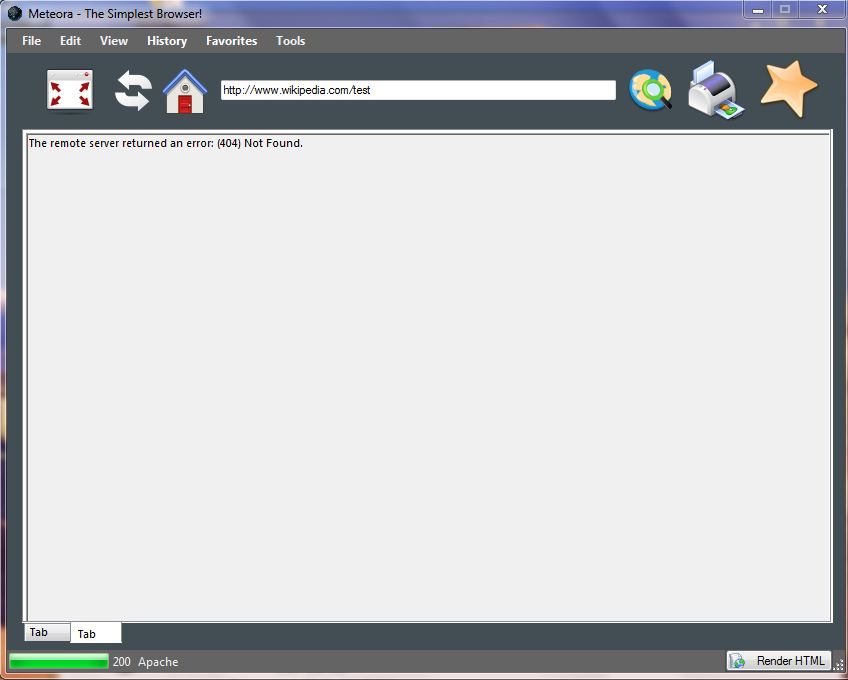


*The new homepage will take effect only after restarting Meteora or when you open a new window, as the homepage is read only during the program startup.* After restarting, Meteora will start with the newly set homepage address.

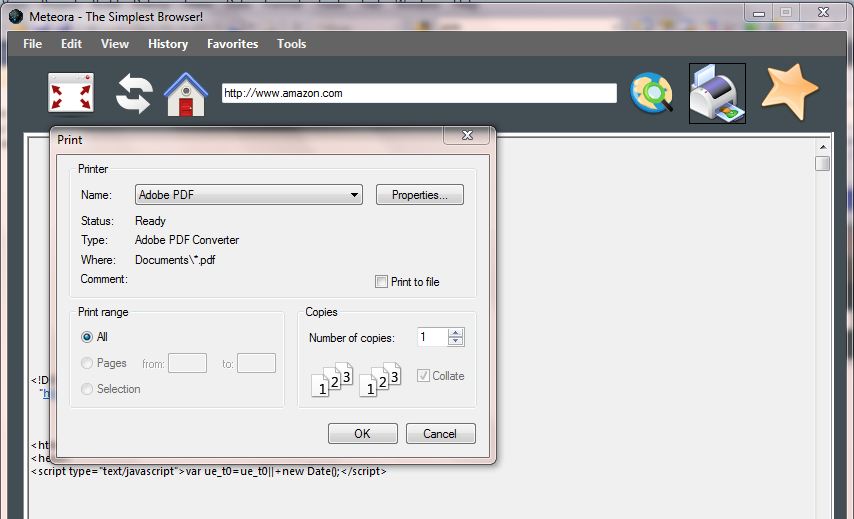


You will receive this message, when there is no internet available.

When there is an error with the http request, the status codes will be displayed as shown below.



Clicking on the print button will open the Print dialog with the standard options as shown in the screenshot below.



This brings us to the end of the developer’s guide for Meteora.

**7. Conclusions**

The highlight of this application is its Simplicity, you have everything you need in front of you without clicking on menus. The full screen mode will put you into the complete web browsing gear and gives you an enhanced browsing experience.

Some of the things I would have done differently with more time, would be, redo the favorites manager and the history, add a context menu for each tabs, reopen closed tabs,..

This project has helped me to get familiarized with Application development in Microsoft Visual Studio 2010. It has helped me to understand how to the various tools, controls and commands already available in visual studio.

This is the first application I have created in my life. I was familiar with ASP.NET, but working with forms was completely different and this project has helped me get familiarized with it and gain experience in it.

**8. References**

.Net Snippets, *C# - Simple Web-request with Web-response,* Available:

http://dotnet-snippets.de/dns/simple-web-request-with-web-response-SID215.aspx

[16 October 2012]

Code Call, *C# Tutorial: Reading and Writing XML Files*, Available:

http://forum.codecall.net/topic/58239-c-tutorial-reading-and-writing-xml-files/ [23 October 2012]

Code Project, *Delegates for Dummies*, Available:

http://www.codeproject.com/Articles/20895/Delegates-for-Dummies [25 October 2012]

Code Project, *Loading and Saving XML to and from a TreeView Control,* Available:

http://www.codeproject.com/Articles/12606/Loading-and-Saving-XML-to-and-from-a-TreeView-Cont [24 October 2012]

Code Project, *LINQ to XML*, Available:

http://www.codeproject.com/Articles/24376/LINQ-to-XML [23 October 2012]

Code Project, *The key to multi-threaded Windows Forms UI interaction*, Available:

http://www.codeproject.com/Articles/2083/The-key-to-multi-threaded-Windows-Forms-UI-interac [22 October 2012]

Dream in Code, *Using the BackgroundWorker in C#,* Available:

http://www.dreamincode.net/forums/topic/112547-using-the-backgroundworker-in-c%23/ [27 October 2012]

Express Computer Online, *Understanding Delegates*, Available:

http://www.expresscomputeronline.com/20021216/techspace2.shtml [20 October 2012]

Microsoft Support, *How to populate a TreeView control with XML data in Visual C# 2005 or in Visual C# .NET*, Available: http://support.microsoft.com/kb/317597 [25 October 2012]

Microsoft Support, *Load and Save XML by Using DOM in .NET Framework with Visual C# .NET,* Available: http://support.microsoft.com/kb/317662 [22 October 2012]

MSDN, *HttpWebRequest Class***,** Available:

http://msdn.microsoft.com/en-us/library/system.net.httpwebrequest.aspx [15 October 2012]

MSDN, *How to: Add and Remove Tabs with the Windows Forms TabControl,*Available:

http://msdn.microsoft.com/en-us/library/zb7xae05.aspx#Y0 [15 October 2012]