



# Visual Studio .NET

Source code exported to PDF

<b>Solution name</b>	HTMLParser.sln
<b>Main Project name</b>	WindowsApplication1
<b>Main Project location</b>	C:\Dev\Products\VSNETcodePrint2005 \HTMLParser\WindowsApplication1 \WindowsApplication1.vbproj
<b>Print date</b>	07 October 2007
<b>Author</b>	<b>Joginder S Nahil</b>

---

## Table of Contents

	Page	Line
<b>HTMLParser [Solution]</b>		
HTMLParser [Project]	1	1
bin [PhysicalFolder]	1	1
HTMLParser.dll.snk [PhysicalFile]	1	1
HTMLParser.cs [ProjectItem]	1	1
HTMLParser [Namespace]	1	10
Parser [Class]	1	12
DecodeScript [Function]	6	566
EncodeScript [Function]	6	557
GetTokens [Function]	7	587
Parser [Function]	2	95
PreprocessScript [Function]	5	452
RemoveComments [Function]	2	315
RemoveSGMLComments [Function]	4	387
RemoveWhitespace [Function]	2	294
TokenEntry [Struct]	2	83

```
00001 using System;
00002 using System.IO;
00003 using System.Text;
00004 using System.Collections;
00005 using System.Collections.Specialized;
00006 using System.Diagnostics;
00007 using System.Globalization;
00008
00009
00010
00011 namespace HTMLParser
00012 {
00013     /// <summary>
00014     /// This is the main HTML parser class. I recommend you don't play around too much in here
00015     /// as it's a little fiddly.
00016     ///
00017     /// Bascially, this class will build a tree containing HtmlNode elements.
00018     /// </summary>
00019     public class Parser
00020     {
00021         private static char[] WHITESPACE_CHARS = "\t\r\n".ToCharArray();
00022
00023         public enum TOKEN_KINDS
00024         {
00025             TOKEN_TEXT = 1,
00026             TOKEN_COLLAPSIBLE_TEXT,
00027             TOKEN_COMMENT,
00028             TOKEN_CSS_PROPERTY_NAME,
00029             TOKEN_CSS_PROPERTY_VALUE,
00030             TOKEN_CSS_SELECTOR,
00031             TOKEN_CSS_STRING_VALUE,
00032             TOKEN_DUMMY,
00033             TOKEN_EXCLUDED_CODE,
00034             TOKEN_FUNCTION_BLOCK_START,
00035             TOKEN_HTML_ATTRIBUTE_NAME,
00036             TOKEN_HTML_ATTRIBUTE_VALUE,
00037             TOKEN_HTML_COMMENT,
00038             TOKEN_HTML_ELEMENT_NAME,
00039             TOKEN_HTML_ENTITY,
00040             TOKEN_HTML_OPERATOR,
00041             TOKEN_HTML_SERVER_SIDE_SCRIPT,
00042             TOKEN_HTML_STRING,
00043             TOKEN_HTML_TAG_DELIMITER_START,
00044             TOKEN_HTML_TAG_DELIMITER_END,
00045             TOKEN_HTML_TAG_TEXT,
00046             TOKEN_IDENTIFIER,
00047             TOKEN_KEYWORD,
00048             TOKEN_LINE_NUMBERS,
00049             TOKEN_NAMESPACE,
00050             TOKEN_NUMBER,
00051             TOKEN_OPERATOR,
00052             TOKEN_PLAIN_TEXT,
00053             TOKEN_PREPROCESSOR_KEYWORD,
00054             TOKEN_PROJECT_INFORMATION,
00055             TOKEN_READ_ONLY_REGION,
00056             TOKEN_STRING,
00057             TOKEN_USER_KEYWORD,
00058             TOKEN_VISIBLE_WHITESPACE,
00059             TOKEN_WIZARD_CODE,
00060             TOKEN_XML_DOC_COMMENT,
00061             TOKEN_XML_TAG,
00062             TOKEN_OUTLINES,
00063             TOKEN_CLASS_MODULE_HEADER,
00064             TOKEN_PAGE_HEADER,
```

1 2 3

```

00064 1 2 3 TOKEN_PROCEDURE_HEADER,
00065     TOKEN_TABLE_OF_CONTENTS,
00066     TOKEN_EOL
00067 };
00068
00069
00070 public enum ParseStatus
00071 {
00072     ReadText = 0,
00073     ReadEndTag = 1,
00074     ReadStartTag = 2,
00075     ReadAttributeName = 3,
00076     ReadAttributeValue = 4,
00077     ReadSelector = 5,
00078     ReadProperty = 6,
00079     ReadPropertyValue = 7,
00080     ReadComment = 8
00081 };
00082
00083 public struct TokenEntry
00084 {
00085     public TOKEN_KINDS Kind;
00086     public string Text;
00087
00088     public TokenEntry(TOKEN_KINDS pkind, string pText)
00089     {
00090         Kind = pkind;
00091         Text = pText;
00092     }
00093 }
00094
00095
00096 public Parser()
00097 {
00098
00099
00100     [The main parser]
00291
00292     #region HTML clean-up functions
00293
00294
00295     /// <summary>
00296     /// This will remove redundant whitespace from the string
00297     /// </summary>
00298     /// <param name="input"></param>
00299     /// <returns></returns>
00300     private string RemoveWhitespace(string input)
00301     {
00302         string output = input.Replace("\r", "");
00303         output = output.Replace("\n", "");
00304         output = output.Replace("\t", " ");
00305         output = output.Trim();
00306         return output;
00307     }
00308
00309     /// <summary>
00310     /// This will remove all HTML comments from the input string. This will
00311     /// not remove comment markers from inside tag attribute values.
00312     /// </summary>
00313     /// <param name="input">Input HTML containing comments</param>
00314     /// <returns>HTML containing no comments</returns>
00315

```

1 2

```
00316 public string RemoveComments(string input)
00317 {
00318     StringBuilder output = new StringBuilder();
00319     int i = 0;
00320     bool inTag = false;
00321
00322     while (i < input.Length)
00323     {
00324         if (i + 4 < input.Length && input.Substring(i, 4).Equals("<!--"))
00325         {
00326             i += 4;
00327             i = input.IndexOf("-->", i);
00328             if (i == -1)
00329             {
00330                 break;
00331             }
00332             i += 3;
00333         }
00334         else if (input.Substring(i, 1).Equals("<"))
00335         {
00336             inTag = true;
00337             output.Append("<");
00338             i++;
00339         }
00340         else if (input.Substring(i, 1).Equals(">"))
00341         {
00342             inTag = false;
00343             output.Append(">");
00344             i++;
00345         }
00346         else if (input.Substring(i, 1).Equals("\\"") && inTag)
00347         {
00348             int string_start = i;
00349             i++;
00350             i = input.IndexOf("\\", i);
00351             if (i == -1)
00352             {
00353                 break;
00354             }
00355             i++;
00356             output.Append(input.Substring(string_start, i - string_start));
00357         }
00358         else if (input.Substring(i, 1).Equals("\'") && inTag)
00359         {
00360             int string_start = i;
00361             i++;
00362             i = input.IndexOf("'", i);
00363             if (i == -1)
00364             {
00365                 break;
00366             }
00367             i++;
00368             output.Append(input.Substring(string_start, i - string_start));
00369         }
00370         else
00371         {
00372             output.Append(input.Substring(i, 1));
00373             i++;
00374         }
00375     }
00376
00377     return output.ToString();
00378 }
00379
```

```

00380 1 2  // <summary>
00381  // This will remove all HTML comments from the input string. This will
00382  // not remove comment markers from inside tag attribute values.
00383  // </summary>
00384  // <param name="input">Input HTML containing comments</param>
00385  // <returns>HTML containing no comments</returns>
00386
00387
00388  private string RemoveSGMLComments(string input)
00389  {
00390      StringBuilder output = new StringBuilder();
00391
00392      int i = 0;
00393      bool inTag = false;
00394
00395      while (i < input.Length)
00396      {
00397          if (i + 2 < input.Length && input.Substring(i, 2).Equals("<!--"))
00398          {
00399              i += 2;
00400              i = input.IndexOf(">", i);
00401              if (i == -1)
00402              {
00403                  break;
00404              }
00405              i += 3;
00406          }
00407          else if (input.Substring(i, 1).Equals("<"))
00408          {
00409              inTag = true;
00410              output.Append("<");
00411              i++;
00412          }
00413          else if (input.Substring(i, 1).Equals(">"))
00414          {
00415              inTag = false;
00416              output.Append(">");
00417              i++;
00418          }
00419          else if (input.Substring(i, 1).Equals("\\") && inTag)
00420          {
00421              int string_start = i;
00422              i++;
00423              i = input.IndexOf("\\", i);
00424              if (i == -1)
00425              {
00426                  break;
00427              }
00428              i++;
00429              output.Append(input.Substring(string_start, i - string_start));
00430          }
00431          else if (input.Substring(i, 1).Equals("\\") && inTag)
00432          {
00433              int string_start = i;
00434              i++;
00435              i = input.IndexOf("\\", i);
00436              if (i == -1)
00437              {
00438                  break;
00439              }
00440              i++;
00441              output.Append(input.Substring(string_start, i - string_start));
00442          }
00443          else
00444          {

```

```

00444 1 2 3 4 5 output.Append(input.Substring(i, 1));
00445      i++;
00446      }
00447  }
00448
00449      return output.ToString();
00450  }
00451
00452


---


00453  /// <summary>
00454  /// This will encode the scripts within the page so they get passed through the
00455  /// parser properly. This is due to some people using comments protect the script
00456  /// and others who don't. It also takes care of issues where the script itself has
00457  /// HTML comments in (in strings, for example).
00458  /// </summary>
00459  /// <param name="input">The HTML to examine</param>
00460  /// <returns>The HTML with the scripts marked up differently</returns>
00461  public string PreprocessScript(string input, string tag_name)
00462  {
00463      StringBuilder output = new StringBuilder();
00464      int index = 0;
00465      int tag_name_len = tag_name.Length;
00466      while (index < input.Length)
00467      {
00468          bool omit_body = false;
00469          if (index + tag_name_len + 1 < input.Length && input.Substring(index, tag_name_len + 1).ToLower().
    »      Equals("<" + tag_name))
00470          {
00471              // Look for the end of the tag (we pass the attributes through as normal)
00472              do
00473              {
00474                  if (index >= input.Length)
00475                  {
00476                      break;
00477                  }
00478                  else if (input.Substring(index, 1).Equals(">"))
00479                  {
00480                      output.Append(">");
00481                      index++;
00482                      break;
00483                  }
00484                  else if (index + 1 < input.Length && input.Substring(index, 2).Equals("/>"))
00485                  {
00486                      output.Append("/>");
00487                      index += 2;
00488                      omit_body = true;
00489                      break;
00490                  }
00491                  else if (input.Substring(index, 1).Equals("\\"))
00492                  {
00493                      output.Append("\\"");
00494                      index++;
00495                      while (index < input.Length && !input.Substring(index, 1).Equals("\\"))
00496                      {
00497                          output.Append(input.Substring(index, 1));
00498                          index++;
00499                      }
00500                  if (index < input.Length)
00501                  {
00502                      index++;
00503                      output.Append("\\"");
00504                  }
00505              }
00506          }
          else if (input.Substring(index, 1).Equals("\\"))

```

```

00507 1 2 3 4 5 6 {
00508     output.Append("\");
00509     index++;
00510     while (index < input.Length && !input.Substring(index, 1).Equals("\"))
00511     {
00512         output.Append(input.Substring(index, 1));
00513         index++;
00514     }
00515     if (index < input.Length)
00516     {
00517         index++;
00518         output.Append("\");
00519     }
00520     }
00521     else
00522     {
00523         output.Append(input.Substring(index, 1));
00524         index++;
00525     }
00526     } while (true);
00527     if (index >= input.Length) break;
00528     // Phew! Ok now we are reading the script body
00529
00530     if (!omit_body)
00531     {
00532         StringBuilder script_body = new StringBuilder();
00533         while (index + tag_name_len + 3 < input.Length && !input.Substring(index, tag_name_len + 3).
00534             >> ToLower().Equals("<" + tag_name + ">"))
00535         {
00536             script_body.Append(input.Substring(index, 1));
00537             index++;
00538         }
00539         // Done - now encode the script
00540         output.Append(EncodeScript(script_body.ToString()));
00541         output.Append("<" + tag_name + ">");
00542         if (index + tag_name_len + 3 < input.Length)
00543         {
00544             index += tag_name_len + 3;
00545         }
00546     }
00547     else
00548     {
00549         output.Append(input.Substring(index, 1));
00550         index++;
00551     }
00552     }
00553     return output.ToString();
00554 }
00555
00556
00557
00558 private static string EncodeScript(string script)
00559 {
00560     string output = script.Replace("<", "[MIL-SCRIPT-LT]");
00561     output = output.Replace(">", "[MIL-SCRIPT-GT]");
00562     output = output.Replace("\r", "[MIL-SCRIPT-CR]");
00563     output = output.Replace("\n", "[MIL-SCRIPT-LF]");
00564     return output;
00565 }
00566
00567 private static string DecodeScript(string script)
00568 {
00569     string output = script.Replace("[MIL-SCRIPT-LT]", "<");

```

```

00569 1 2 3 output = output.Replace("[MIL-SCRIPT-GT]", ">");
00570 output = output.Replace("[MIL-SCRIPT-CR]", "\r");
00571 output = output.Replace("[MIL-SCRIPT-LF]", "\n");
00572 return output;
00573 }
00574
00575 #endregion
00576
00577 #region HTML tokeniser
00578
00579 /// <summary>
00580 /// This will tokenise the HTML input string.
00581 /// </summary>
00582 /// <param name="input"></param>
00583 /// <returns></returns>
00584 /// [
00585 ///
00586
00587


---


00588 public ArrayList GetTokens(string input,
00589 ref ParseStatus status)
00590 {
00591     ArrayList TokenArray = new ArrayList();
00592     int i = 0;
00593     string token;
00594
00595     Debug.WriteLine "[" + status.ToString()
00596         + "] [" + input + "];"
00597
00598     while (i < input.Length)
00599     {
00600         if (status == ParseStatus.ReadText)
00601         {
00602
00603             // We enter this state when we are not inside an element
00604             // for example we are not inside < >
00605
00606             if (i + 2 < input.Length && input.Substring(i, 2).Equals("</"))
00607             {
00608                 // *****
00609                 // Input starts with </
00610                 // *****
00611
00612                 i += 2;
00613
00614                 TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END, "</"));
00615                 status = ParseStatus.ReadEndTag;
00616             }
00617
00618             else if (input.Substring(i, 1).Equals("<"))
00619             {
00620                 /*
00621                 *****
00622                 Input starts with <
00623                 *****
00624                 */
00625
00626                 TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_START, "<"));
00627                 i++;
00628
00629                 status = ParseStatus.ReadStartTag;
00630             }
00631             else
00632             {

```

```

00633 1 2 3 4 5 6  /*
00634  /*
00635  Find the next <
00636  /*
00637  */
00638  int next_index = input.IndexOf("<", i);
00639
00640  if (next_index == -1)
00641  {
00642  /*
00643  /*
00644  Save < as a token
00645  /*
00646  */
00647  TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_START, input.
    »      Substring(i)));
00648  break;
00649  }
00650  else
00651  {
00652  /*
00653  /*
00654  Everything before < must be Text token
00655  /*
00656  */
00657  TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_TEXT, input.Substring(i, next_index - i)));
00658
00659  i = next_index;
00660  }
00661  }
00662  }
00663
00664
00665
00666  else if (status == ParseStatus.ReadComment )
00667  {
00668  /*
00669  /*
00670  We are in the middle of reading comment block
00671  /*
00672  */
00673
00674  bool inString = false;
00675  string Temp = "";
00676
00677  while (i < input.Length && input.Substring(i, 1).IndexOfAny("\r".ToCharArray()) == -1)
00678  {
00679  if (input.Substring(i, 1).Equals("\\"))
00680  {
00681  //We have a quote " character
00682  Temp = Temp + input.Substring(i, 1);
00683
00684  i++;
00685  inString = !inString;
00686  }
00687  else if (i + 3 <= input.Length && input.Substring(i, 3).Equals("-->"))
00688  {
00689
00690  if (inString)
00691  {
00692  Temp = Temp + input.Substring(i, 3);
00693  i += 3;
00694  }
00695  else
00696  {

```

```

00697 1 2 3 4 5 6 7 8 status = ParseStatus.ReadText;
00698      break;
00699      }
00700    }
00701    else
00702    {
00703      Temp = Temp + input.Substring(i, 1);
00704      i++;
00705    }
00706  }
00707
00708  if (Temp != "")
00709  {
00710    TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_COMMENT, Temp));
00711    Temp = "";
00712  }
00713
00714  if (status == ParseStatus.ReadText)
00715  {
00716    TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_COMMENT, input.Substring(i, 2)));
00717    // TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END ,input
    //                               t.Substring(i+2,1) ));
00718    i += 2;
00719  }
00720  }
00721
00722
00723
00724
00725  else if (status == ParseStatus.ReadStartTag)
00726  {
00727    // Skip leading whitespace before the Element/Tag name
00728    while (i < input.Length && input.Substring(i, 1).IndexOfAny(WHITESPACE_CHARS) != -1)
00729    {
00730      i++;
00731    }
00732
00733    // *****
00734    // Find the end of Element/Tag name
00735    // *****
00736
00737    int tag_name_start = i;
00738    while (i < input.Length && input.Substring(i, 1).IndexOfAny("\r\n\t/>".ToCharArray()) == -1)
00739    {
00740      i++;
00741    }
00742
00743    token = input.Substring(tag_name_start, i - tag_name_start);
00744
00745    if (token.StartsWith("!"))
00746    {
00747      /*
00748      *****
00749      * All elements are colored with their defined fonts and
00750      * colors, however, comment tag has be colored comment
00751      * color. So we tag it as 'Comment'
00752      *****
00753      */
00754
00755      TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_COMMENT, token.Substring(0, 1)));
00756
00757      if (token.StartsWith("!-"))
00758      {
00759        /*
00760        *****

```

```

00761 1 2 3 4 5 6 7 * Save all the text between <!-- --> as one token, however
00762 * there can be --> inside quotes before terminating -->
00763 *
00764 * Example:
00765 *
00766 * <!-- "Hello -->" -->
00767 * This may be illegal but I have seen it used so we need
00768 * to remember the text between "" as a separate token
00769 *****
00770 */
00771
00772 status = ParseStatus.ReadComment;
00773
00774 TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_COMMENT, token.Substring(1, token.Length
    > - 1)));
00775
00776 bool inString = false;
00777 string Temp = "";
00778
00779 while (i < input.Length && input.Substring(i, 1).IndexOfAny("\r".ToCharArray()) == -1)
00780 {
00781     if (input.Substring(i, 1).Equals("\""))
00782     {
00783         //We have a quote " character
00784         Temp = Temp + input.Substring(i, 1);
00785
00786         i++;
00787         inString = !inString;
00788     }
00789     else if (i + 3 <= input.Length && input.Substring(i, 3).Equals("-->"))
00790     {
00791
00792         if (inString)
00793         {
00794             Temp = Temp + input.Substring(i, 3);
00795             i += 3;
00796         }
00797         else
00798         {
00799             status = ParseStatus.ReadText;
00800             break;
00801         }
00802     }
00803     else
00804     {
00805         Temp = Temp + input.Substring(i, 1);
00806         i++;
00807     }
00808 }
00809
00810 if (Temp != "")
00811 {
00812     TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_COMMENT, Temp));
00813     Temp = "";
00814 }
00815
00816 if (status == ParseStatus.ReadText)
00817 {
00818     TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_COMMENT, input.Substring(i, 2)));
00819     // TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END ,input
    > t.Substring(i+2,1) ));
00820     i += 2;
00821 }
00822 }
00823 1 2 3 4 5 6

```

```

00824 1 2 3 4 5 6 else
00825 {
00826 // Must be an AttributeName token
00827 TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_ATTRIBUTE_NAME, token.Substring(1
    » , token.Length - 1)));
00828 }
00829 }
00830 }
00831 }
00832 else
00833 {
00834 // Must be an Element/Tag token
00835 TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_ELEMENT_NAME, token));
00836 }
00837 }
00838 }
00839 // Skip trailing whitespace in tag
00840 while (i < input.Length && input.Substring(i, 1).IndexOfAny(WHITESPACE_CHARS) != -1)
00841 {
00842 i++;
00843 }
00844 }
00845 /*
00846 * *****
00847 * Is it terminator />
00848 * *****
00849 */
00850 if (i + 1 < input.Length && input.Substring(i, 1).Equals("/>"))
00851 {
00852 TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END, "/>"));
00853 }
00854 status = ParseStatus.ReadText;
00855 i += 2;
00856 }
00857 }
00858 /*
00859 * *****
00860 * Is it terminator >
00861 * *****
00862 */
00863 }
00864 else if (i < input.Length && input.Substring(i, 1).Equals(">"))
00865 {
00866 TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END, ">"));
00867 }
00868 status = ParseStatus.ReadText;
00869 i++;
00870 }
00871 /*
00872 * *****
00873 * Is it a start of new element <
00874 * *****
00875 */
00876 }
00877 else if (i < input.Length && input.Substring(i, 1).Equals("<"))
00878 {
00879 // status remains unchanged
00880 }
00881 else
00882 {
00883 if (status != ParseStatus.ReadComment )
00884 {
00885 status = ParseStatus.ReadAttributeName;
00886 }
00887 }

```

```

00888 1 2 3 4 5 }
00889
00890 /*
00891  * *****
00892  * Looking for tag/element name after </xxxxxx>
00893  * *****
00894  */
00895
00896 else if (status == ParseStatus.ReadEndTag)
00897 {
00898     // Skip leading whitespace in tag
00899     while (i < input.Length && input.Substring(i, 1).IndexOfAny(WHITESPACE_CHARS) != -1)
00900     {
00901         i++;
00902     }
00903     // Read closing tag name
00904     int tag_name_start = i;
00905     while (i < input.Length && input.Substring(i, 1).IndexOfAny("\r\n\t>".ToCharArray()) == -1)
00906     {
00907         i++;
00908     }
00909
00910     TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_ELEMENT_NAME, input.Substring(
00911         tag_name_start, i - tag_name_start)));
00912
00913     // Skip trailing whitespace in tag
00914     while (i < input.Length && input.Substring(i, 1).IndexOfAny(WHITESPACE_CHARS) != -1)
00915     {
00916         i++;
00917     }
00918     if (i < input.Length && input.Substring(i, 1).Equals(">"))
00919     {
00920         TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END, ">"));
00921         status = ParseStatus.ReadText;
00922         i++;
00923     }
00924     else if (status == ParseStatus.ReadAttributeName)
00925     {
00926         /*
00927          * *****
00928          * Read Attribute Name
00929          * *****
00930          */
00931
00932         while (i < input.Length && input.Substring(i, 1).IndexOfAny(WHITESPACE_CHARS) != -1)
00933         {
00934             i++;
00935         }
00936         int attribute_name_start = i;
00937
00938         while (i < input.Length && input.Substring(i, 1).IndexOfAny("\r\n\t/>=".ToCharArray()) == -1)
00939         {
00940             i++;
00941         }
00942         string AttributeName = "";
00943
00944         AttributeName = input.Substring(attribute_name_start, i - attribute_name_start);
00945
00946         // If an attribute starts with " then search for matching "
00947
00948         if (AttributeName.Length >= 1)
00949         {
00950             if (AttributeName.Substring(0, 1).Equals("\""))
00951             {

```

```

00952 1 2 3 4 5 6 7 while (i < input.Length)
00953 {
00954     if (input.Substring(i, 1).Equals("\\"))
00955     {
00956         i++;
00957         break;
00958     }
00959
00960     i++;
00961 }
00962 }
00963 }
00964
00965     AttributeName = input.Substring(attribute_name_start, i - attribute_name_start);
00966     if (AttributeName.StartsWith("\\"))
00967     {
00968         TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_STRING, AttributeName));
00969     }
00970
00971     else
00972     {
00973         TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_ATTRIBUTE_NAME, AttributeName));
00974     }
00975     while (i < input.Length && input.Substring(i, 1).IndexOfAny(WHITESPACE_CHARS) != -1)
00976     {
00977         i++;
00978     }
00979     if (i + 1 < input.Length && input.Substring(i, 2).Equals("/>"))
00980     {
00981         TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END, "/>"));
00982         status = ParseStatus.ReadText();
00983         i += 2;
00984     }
00985
00986     else if (i < input.Length && input.Substring(i, 1).Equals(">"))
00987     {
00988         TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END, ">"));
00989         status = ParseStatus.ReadText();
00990         i++;
00991     }
00992
00993     else if (i < input.Length && input.Substring(i, 1).Equals("="))
00994     {
00995         TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_OPERATOR, "="));
00996         i++;
00997         status = ParseStatus.ReadAttributeValue();
00998     }
00999     else if (i < input.Length && input.Substring(i, 1).Equals("/"))
01000     {
01001         i++;
01002     }
01003 }
01004 else if (status == ParseStatus.ReadAttributeValue)
01005 {
01006     /*
01007     * *****
01008     * Read Attribute Value i.e Attribute = xxxxxxxx
01009     * *****
01010     */
01011
01012     while (i < input.Length && input.Substring(i, 1).IndexOfAny(WHITESPACE_CHARS) != -1)
01013     {
01014         i++;
01015     }
01016     if (i < input.Length && input.Substring(i, 1).Equals("\\"))

```

```

01017 1 2 3 4 5 {
01018     int value_start = i;
01019     i++;
01020     while (i < input.Length && !input.Substring(i, 1).Equals("\\"))
01021     {
01022         i++;
01023     }
01024     if (i < input.Length && input.Substring(i, 1).Equals("\\"))
01025     {
01026         i++;
01027     }
01028     TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_ATTRIBUTE_VALUE, input.Substring(
01029         value_start, i - value_start)));
01030     >
01031     status = ParseStatus.ReadAttributeName;
01032 }
01033
01034 else if (i < input.Length && input.Substring(i, 1).Equals("\\"))
01035 {
01036     int value_start = i;
01037     i++;
01038     while (i < input.Length && !input.Substring(i, 1).Equals("\\"))
01039     {
01040         i++;
01041     }
01042     if (i < input.Length && input.Substring(i, 1).Equals("\\"))
01043     {
01044         i++;
01045     }
01046     TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END, input.Substring(
01047         value_start + 1, i - value_start - 2)));
01048     >
01049     status = ParseStatus.ReadAttributeName;
01050 }
01051 else
01052 {
01053     int value_start = i;
01054     while (i < input.Length && input.Substring(i, 1).IndexOfAny("\r\n\t/>".ToCharArray()) == -1)
01055     {
01056         i++;
01057     }
01058     TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END, input.Substring(
01059         value_start, i - value_start)));
01060     >
01061     while (i < input.Length && input.Substring(i, 1).IndexOfAny(WHITESPACE_CHARS) != -1)
01062     {
01063         i++;
01064     }
01065     status = ParseStatus.ReadAttributeName;
01066 }
01067 while (i < input.Length && input.Substring(i, 1).IndexOfAny(WHITESPACE_CHARS) != -1)
01068 {
01069     i++;
01070 }
01071
01072 if (i + 1 < input.Length && input.Substring(i, 2).Equals("/>"))
01073 {
01074     TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END, "/>"));
01075     status = ParseStatus.ReadText;
01076     i += 2;
01077 }
01078 else if (i < input.Length && input.Substring(i, 1).Equals(">"))

```

---

```
01079 1 2 3 4 5 {
01080     TokenArray.Add(new TokenEntry(TOKEN_KINDS.TOKEN_HTML_TAG_DELIMITER_END, ">"));
01081     i++;
01082     status = ParseStatus.ReadText;
01083 }
01084 }
01085 }
01086 }
01087
01088     return TokenArray;
01089 }
01090 }
01091
01092     #endregion
01093 }
01094 }
01095 }
01096 }
```

## Index

Add	7, 9, 10, 12-14	TOKEN_COMMENT	1, 9, 10
Append	3-6	TOKEN_CSS_PROPERTY_NAME	1
ArrayList	7	TOKEN_CSS_PROPERTY_VALUE	1
attribute_name_start	12	TOKEN_CSS_SELECTOR	1
AttributeName	2, 11, 12, 14	TOKEN_CSS_STRING_VALUE	1
Collections	1	TOKEN_DUMMY	1
Debug	7	TOKEN_EOL	2
DecodeScript	6	TOKEN_EXCLUDED_CODE	1
Diagnostics	1	TOKEN_FUNCTION_BLOCK_START	1
EncodeScript	6	TOKEN_HTML_ATTRIBUTE_NAME	1, 11, 13
Equals	3-5, 7, 8, 10-14	TOKEN_HTML_ATTRIBUTE_VALUE	1, 14
GetTokens	7	TOKEN_HTML_COMMENT	1
Globalization	1	TOKEN_HTML_ELEMENT_NAME	1, 11, 12
HTMLParser	1, 2, 4, 5, 7, 8, 10, 11, 13, 14	TOKEN_HTML_ENTITY	1
i	1-14	TOKEN_HTML_OPERATOR	1
index	3-6, 8, 9, 11-13	TOKEN_HTML_SERVER_SIDE_SCRIPT	1
IndexOf	3, 4, 8, 9, 11-13	TOKEN_HTML_STRING	1
IndexOfAny	8, 10, 12-14	TOKEN_HTML_TAG_DELIMITER_END	1, 7, 11-14
input	2-5, 7-14	TOKEN_HTML_TAG_DELIMITER_START	1, 7
inString	8, 10	TOKEN_HTML_TAG_TEXT	1
inTag	3, 4	TOKEN_IDENTIFIER	1
IO	1, 2, 7, 15	TOKEN_KEYWORD	1
Kind	1, 2, 7, 9, 10, 12-14	TOKEN_KINDS	1, 2, 7, 9, 10, 12-14
Length	3-8, 10-14	TOKEN_LINE_NUMBERS	1
next_index	8	TOKEN_NAMESPACE	1
omit_body	5	TOKEN_NUMBER	1
output	2-4, 6, 7	TOKEN_OPERATOR	1, 13
Parser	1-3, 5, 6, 8, 9, 11, 12, 14, 15	TOKEN_OUTLINES	1
ParseStatus	2, 7, 8, 10-14	TOKEN_PAGE_HEADER	1
pkind	2	TOKEN_PLAIN_TEXT	1
PreprocessScript	5	TOKEN_PREPROCESSOR_KEYWORD	1
pText	2	TOKEN_PROCEDURE_HEADER	2
ReadAttributeName	2, 11, 14	TOKEN_PROJECT_INFORMATION	1
ReadAttributeValue	2, 13	TOKEN_READ_ONLY_REGION	1
ReadComment	2, 8, 10, 11	TOKEN_STRING	1, 13
ReadEndTag	2, 7, 12	TOKEN_TABLE_OF_CONTENTS	2
ReadProperty	2	TOKEN_TEXT	1, 8
ReadPropertyValue	2	TOKEN_USER_KEYWORD	1
ReadSelector	2	TOKEN_VISIBLE_WHITESPACE	1
ReadStartTag	2, 7, 9	TOKEN_WIZARD_CODE	1
ReadText	2, 7, 9-11, 13, 14	TOKEN_XML_DOC_COMMENT	1
RemoveComments	3	TOKEN_XML_TAG	1
RemoveSGMLComments	4	TokenArray	7-9, 11-14
RemoveWhitespace	2	TokenEntry	2, 7, 9, 10, 12-14
Replace	2, 6	ToLower	5, 6
script	1, 5, 6	ToString	3, 5, 6
script_body	6	Trim	2
Specialized	1	value_start	14
StartsWith	9, 13	WHITESPACE_CHARS	1, 9, 11-13
status	2, 7, 8, 10-14	WriteLine	7
string_start	3, 4		
StringBuilder	3-6		
Substring	3-10, 12-14		
System	1		
tag_name	5, 6, 9, 12		
tag_name_len	5, 6		
tag_name_start	9, 12		
Temp	8, 10		
Text	1, 2, 7, 9-11, 13, 14		
ToCharArray	1, 8, 10, 12, 14		
token	1, 2, 7-10, 12-14		
TOKEN_CLASS_MODULE_HEADER	1		
TOKEN_COLLAPSIBLE_TEXT	1		