“OVERVIEW OF THE DOCUMENT OBJECT MODEL (DOM), A.K.A. DHTML UNDER WINDOWS”

Rony G. Flatscher

Department of Management Information Systems/Business Informatics at the

Vienna University of Economics and Business Administration (Austria)

ABSTRACT

With the ubiquitity of the World Wide Web (WWW), its document format "Hypertext Markup Language" (HTML) and its planned successor „xhtml“ definded with the "Extensible Markup Language" (XML) have become so important that interacting with applications processing such text became a necessity. In the past years the World Wide Web consortium has created a set of application programming interfaces which allow programmers to manipulate the structure and content of HTML and/or XML files. In the case of WWW browsers, this allows for creating programmatically dynamic interfaces which can get changed on the fly while the user sits in front of the browser displaying those files and/or interfacing with the browser via the keyboard and mouse: the "Document Object Model" (DOM), dubbed "dynamic HTML" (DHTML) by Microsoft.

This paper gives a conceptual overview of the architectural principles of HTML/XML and the role of "Cascading Style Sheets" (CSS) in this context. Building on this foundation, the DOM is introduced and Object Rexx examples under Microsoft's Internet Explorer (a Windows Scripting Host application) are demonstrated to stress some of the most important features of DOM, like generating text on the fly or creating programs which react upon user events like mouse clicks.

1 MARKUP LANGUAGES

The World Wide Web (WWW) has made a tagging language ubiquituos which is called „Hypertext Markup Language“ (HTML, cf. [W3HTML]). The principles and rules of this markup („tagging“) language have been defined with the means of an ANSI and international ISO/IETC standard, the Standardized Markup Language (SGML).

1.1 Tags and Elements

Markup languages allow to enclose portions of text in tags. This process is called „marking up text“. For each opening tag there must be a matching closing tag:

- An „opening tag“ is represented by an opening angle bracket, a (tag) name, optional attributes and a closing angle bracket, e.g. <some_tag_name>, and
- A „closing tag“ is represented by an opening angle bracket, a slash, a matching (tag) name and a closing angle bracket, e.g. </some_tag_name>.

An „element“ in this context is the sequence „opening tag“, content (usually text) and „closing tag“, i.e. the tags themselves are part of an element. Tags must not overlap, but may contain additional tags within, which makes up a hierarchy. Therefore one can always determine the root (the very first element) of marked up text, and starting out from the root one may process each subsequent (child) node, effectively walking the entire tree. It is rather easy to devise programs which are able to parse text according to the markup and create a resulting (parse) tree, in which there exists a node for each element.

Opening tags may carry additional information, if attributes in the form of „`attributeName`“ equal sign and the value enclosed in double quotes exist.[1)] Every element may possess an attribute value to uniquely identify an element within a document, if the attribute's name is called '`id`'[2)]. If there are elements which the

---

1) In XML the enclosing of attribute values with quotes is mandatory and the quote delimiters can be single quotes in addition to double quotes.

2) The precursor of the 'ID' attribute was called 'NAME' and is used in the context of the 'A'-element.

author thinks belong together for one reason or another, then the attribute value for an attribute named ‘`class`’ should include a string denominating that class name.

1.2 The Document Type Definition (DTD)

Document Type Definitions (DTD) define markup languages: they allow for denoting exaclty the names of available tags, whether tags may carry additional information in the form of attributes within the opening tag, in which case the allowable attributes and their types must be defined. In addition, DTD define a „content model“, which is a set of rules determining how many times and in which sequence tags may be applied to text.

Document Type Definitions (DTD) are usually defined in the Standard General Markup Language (SGML) language or with the Extensible Markup Language (XML, cf. [W3XML]), which can be regarded as (rather large) subset of SGML.

1.3 „Instance“ of a DTD

A document containing marked up text is called an „instance“ of some Document Type Definition (DTD), if one used the rules of that DTD to markup the text. There are parsers which are able to read the content of DTDs and check the instance, whether the mark ups occur according to the rules, which are laid out in the DTD. If such a check asserts that all DTD mark up rules were applied correctly, then one can state that the instance is „valid“, i.e. it represents a „valid document“.

1.4 The Hypertext Markup Language (HTML)

The Hypertext Markup Language (HTML) was devised together with the World Wide Web (WWW) to easily allow plain text to be marked up with information, such as simple links to other documents, text which should serve as headers, or paragraphs and the like. The World Wide Web Consortium (W3C) has standardized and frozen HTML at level 4.01. The DTD for HTML is expressed in original SGML, which among other things allows for the following (in this context) most-notable features:

```
<html>
  <head>
     <title>This is my HTML file</title>
  </head>
  <body>
     <h1>Important Heading</h1>
     <p>This <span class="verb">is</span> the
        first paragraph.
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.
     <p id="9876">This <span class="verb">is</span> it.
  </body>
</html>
```

Figure 1: Text with HTML markup.

tag- and attribute names can be given case-indepently, some end-tags may be omitted in places where a program can infer the missing end tags by itself (e.g. a new paragraph ends implicitly a previous one).

Figure 1 depicts text, which has been marked up with HTML, figure 2 shows a possible rendering (formatting) by an HTML browser.

1.5 The Extensible Markup Language (XML)

The Extensible Markup Language (XML) was created by the World Wide Web Consortium (W3C) as slightly simplified version of SGML, making it easier to create

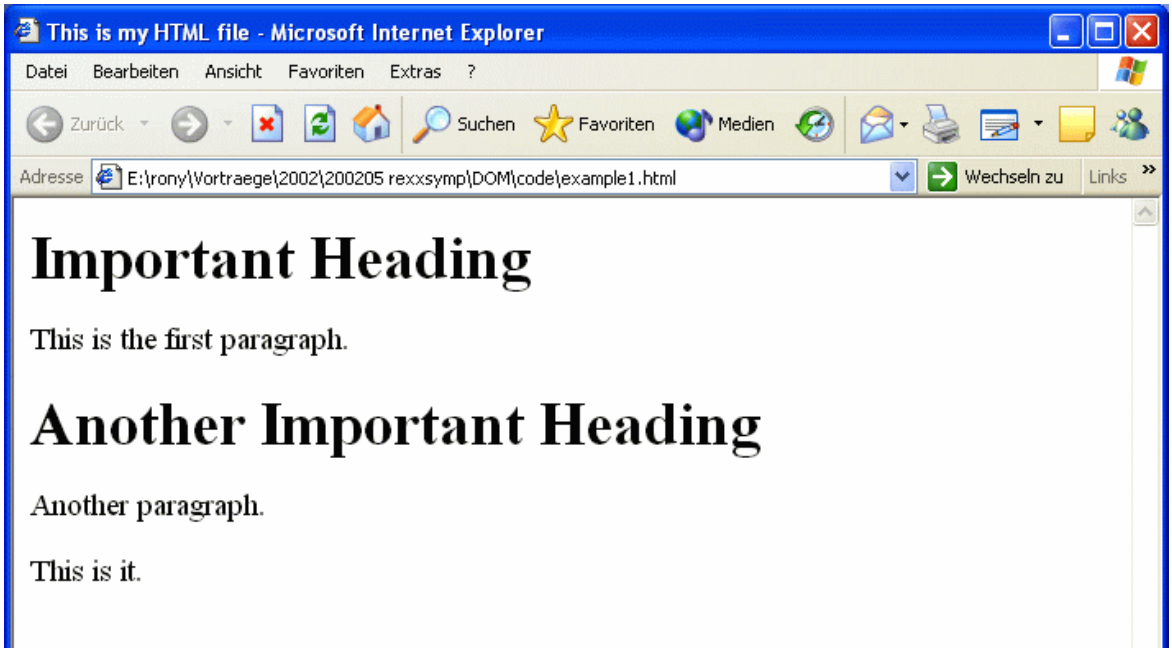

Figure 2: A possible rendering of the HTML text of figure 1.

tools which can implement all of ist features. The most notable features (in this context) are: tag and attribute names are *case sensitive*, end tags must not be omitted, rather end tags must be always supplied, in addition to double quotes attribute values can also be enclosed in single q<uotes (apostrophes). In comparison to SGML there is a short tag form to represent an empty tag: opening angle bracket, tag name, optional attributes, slash and closing angle bracket, e.g. <some_tag_name/>.

As the tagging rules are more rigid than with SGML it becomes possible for creating XML instances (documents containing marked up tags), which do not possess a matching DTD, because it is always possible for an application to create a matching DTD after the XML instance was analyzed.

1.6 Cascading Style Sheets (CSS)

The World Wide Web Consortium (W3C, cf. [W3C]) has devised a standard which allows to define formatting rules for markup languages: "Cascading Style Sheets" (CSS, cf. [W3CSS]). This way the marking up of text adds structural information, which can be exploited by applications, one being the formatting application of WWW browsers.

```
<html>
  <head>
     <title>This is my HTML file</title>
     <link rel="stylesheet" type="text/css" href="example2.css">
  </head>
  <body>
     <h1>Important Heading</h1>
     <p>This <span class="verb">is</span> the
        first paragraph.
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.
     <p id="9876">This <span class="verb">is</span> it.
  </body>
</html>
```

Figure 3: HTML marked up text with a link to a cascading style sheet file (cf. figure 5).

The formatting rules are expressed in the form of text, which usually gets stored in its own files. One is able to define the formatting of certain tags, a specialized

f

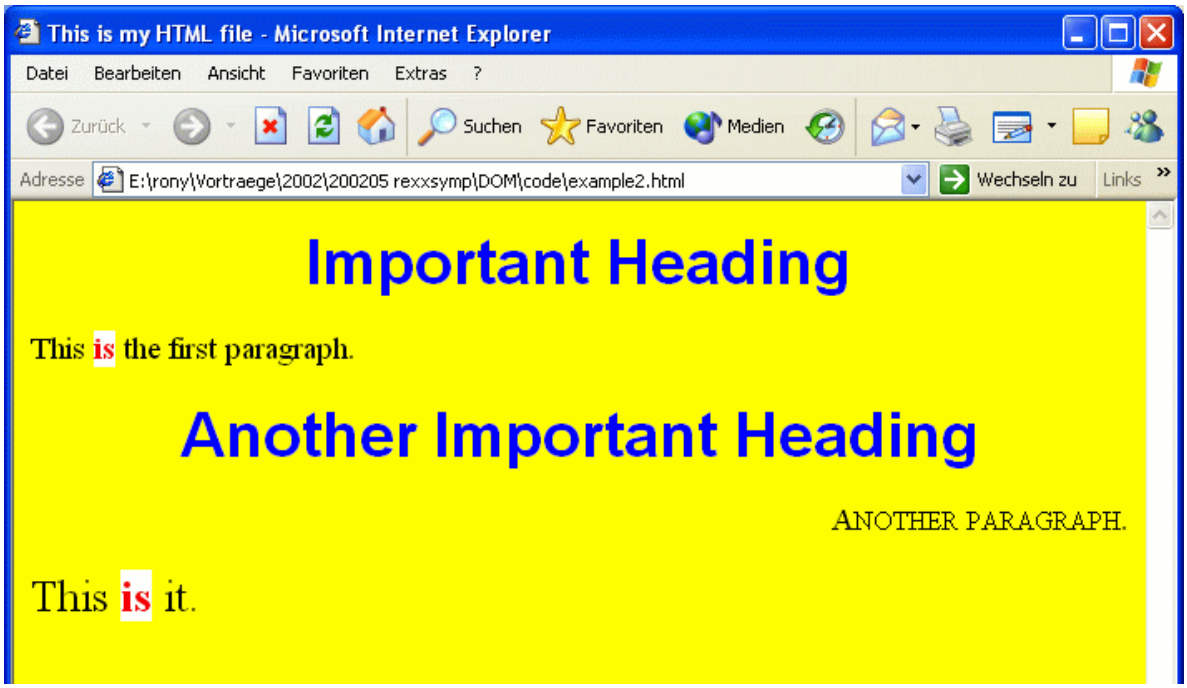

Figure 4: A possible rendering of the HTML text of figure 3.

ormatting depending on a certain sequence of tags (elements), as well as making the formatting dependent on pre-defined values in attributes, most notably the attributes „`class`“ and „`id`“.[3)]

Figure 3 depicts HTML text which adds an empty link element to figure 1, which contains attributes that identify a cascading stylesheet file to be used for formatting. Figure 4 shows a possible rendering and figure 5 displays the content of the cascading style sheet file named „example2.css“.

By comparing figures 2 and 4 one can experience a dramatic difference in appearence which is caused by the usage of a cascading style sheet for formatting the HTML text of figure 3, yielding the rendering of figure 4. A CSS-rule consists of plain text, first giving a match expression followed by formatting definitions to applied to all matching elements, enclosed within curly brackets:

a) „Rule # 1“: The first rule relates to HTML tags named „h1“, which are commonly referred to as „header level 1“ tags. The formatting rule mandates that the text of

[3)] The value of the „class“ attribute may be one or more strings delimited by spaces, the value of the attribute „id“ needs to be a string which is unique within the document where it is defined and therefore serves as a unique identifier.

```
      /* rule # 1: match all tags 'h1' */
h1      { color: blue;
          text-align: center;
          font-family: Arial,sans-serif;
          font-size: 200%; }

      /* rule # 2: match all tags 'body' (there is only one) */
body    { background-color: yellow;
          font-family: Times, Avantgarde;
          font-size: small; }

      /* rule # 3: match all elements possessing the string 'verb'
                   in the value of an attribute named 'class' */
.verb   { background-color: white;
          color: red;
          font-weight: 900; }

      /* rule # 4: match exactly the element, having exactly the
                   value 'xyz1' in an attribute named 'id' */
#xyz1   { font-variant: small-caps;
          text-align: right; }

      /* rule # 5: match exactly the element, having exactly the
                   value '9876' in an attribute named 'id' */
#9876   { font-size: large; }
```

Figure 5: The cascading style sheet named „example2.css“.

such elements has to be formatted with a blue text color, the text has to be centered, the size of the font should be doubled, and if possible the font named „`Arial`“ should be used and in the case that it is not available, just any other font without serifs, like „`Helvetica`“ or the like.

b) „Rule # 2“: This rule determines that the „`body`“ element[4)] has a yellow background, the font size must be small and that the font family „`Times`“ and if not available „`Avantgarde`“ should be used.

c) „Rule # 3“: The third rule mandates, that elements with an attribute named „`class`“ possessing a value of „`verb`“ should be formatted with a red bold font („`font-weight: 900`“) on a white background. If a name is preceded by a dot, then it is assumed to be the explicit value in a „`class`“ attribute.

4) This element represents the entire document rendered in the browser's document window. Therefore ist style determines the „default“ style for the document.

d) „Rule # 4“ and „Rule # 5“: These rules define formatting rules for two different elements, which are identified by their „`id`“ attribute set to „`xyz1`“ and „`9876`“ respectively. If a name is preceded by the pound sign (#), then it is assumed to be a (unique) value of an „`id`“ attribute.

CSS definitions can also be given in the„`head`“ element, enclosed within a tag named „`style`“. In addition one can express CSS formatting rules individually for elements, contained in the value of an element attribute named „`style`“.

2 THE DOCUMENT OBJECT MODEL (DOM)

The „Document Object Model“ (DOM, cf. [W3DOM]) is a standard defined by the World Wide Web consortium for allowing the interaction with and manipulation of HTML- and XML-documents, which have been successfully parsed. The result of this process is organized in a so called „parse tree“ which mirrors the hierarchy of HTML- and XML-documents by representing each element as a node. Figure 6[5)] depicts the HTML tagged file of figure 3 in the form of a parse tree.

WWW-browsers like Microsoft Internet Explorer (MSIE) or AOL Netscape parse HTML- and XML-documents and thereafter apply the formatting rules specified with CSS to the parse tree and usually present the results via the CRT. At this point it becomes possible for users to interact with the rendered documents with the help of a mouse or keyboard. An example of this interaction is the navigation of hyperlinks by choosing them with a mouse click. Upon return from such a chosen hyperlink it is

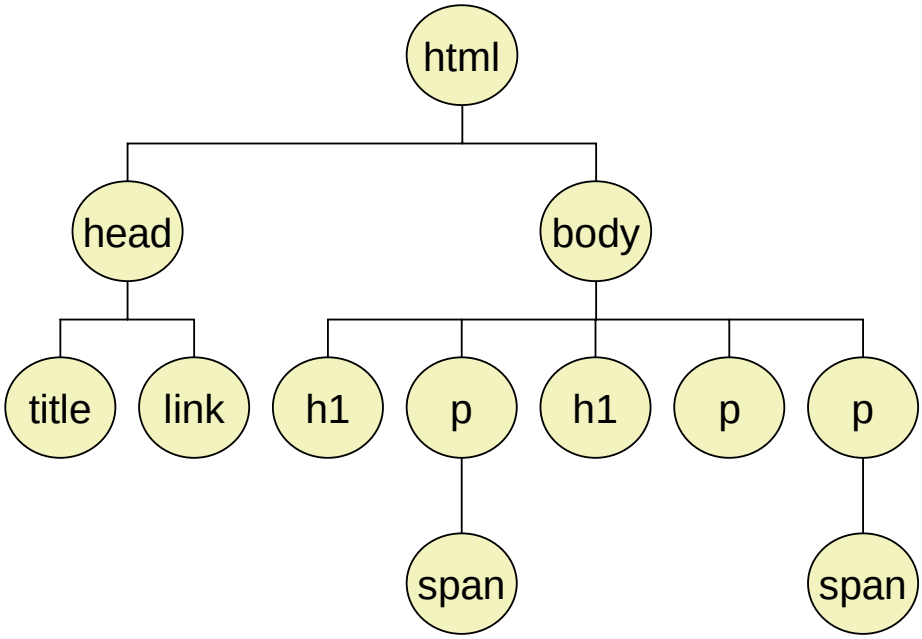

Figure 6: The parse tree of the HTML marked up text of figure 3.

5) Attributes and plain text are not shown as nodes for brevity.

often the case that the visited hyperlink changes its visual representation to indicate to the user, that a particular link has been followed („visited“) already.

The DOM standard defines the set of (language independent) application programming interfaces, which among other things allows the programmer

a) to manipulate the parse tree (e.g. by querying, changing, inserting, deleting elements with its attributes and text) and

b) to react upon events, such as button-presses, mouse- or keyboard-events as well as application related events like "document loaded".

Although DOM has been defined with no specific programming language in mind, it is interesting to note, that for all practical reasons JavaScript (a.k.a. ECMA-Script) has been employed for DOM programming almost exclusively.

2.1 Embedding Programs

Embedding programs is rather straight forward by using the "`script`" tag and determining the name of the programming language with an attribute "language"[6]. The text of this script element should be defined as a "`CDATA`"-section by enclosing it within the strings "`<![CDATA[`" and "`]]>`". Such a "character data" section will cause the HTML parser to not interpret the otherwise special characters < (less than character[7]), > (greater than character) and & (ampersand character). Alternatively, script code can be stored in its own file and can be referred to with an attribute named "src" and a value which is a URL[8].

Any number of script elements can be embedded in the head and body element of the HTML document and will get loaded when the browser loads the document.

---

[6] For the programming language JavaScript/ECMA Script the value should be accorgingly "JavaScript". The Microsoft version of this language is called "JScript".

[7] In HTML- or XML-based mark up this character is represented as the entity "`<`".

[8] It is possible to only give the name of the file in which the code of the script is stored. In this case the usual processing takes place to find the file, comparable to the seeking rules for images and the like.

Whenever a script element is found its code is presented to the language processor which is registered with the browser for further processing, i.e. for its execution. The sequence in which the script elements are processed is the so called "document order", therefore the sequence in which these elements are found in the HTML document.

2.2 Associating Events with Programs

The HTML standards defines a set of attributes which represent events and usually start with the characters "on", like "onclick", "onmouseover" or "onload" and the like. These attributes allow for defining program code as their value and in such a case need an additional attribute "language" with a value that indicates the programming language used.

3 THE MICROSOFT INTERNET EXPLORER (MSIE)

Microsoft has been developing its own WWW browser, named "Microsoft Internet Explorer" (MSIE). This browser got enhanced with the Windows Script Host (WSH) capability, allowing for easy scripting in that environment. The WSH architecture builds on OLE/ActiveX automation and allows among other things an application to pre-register OLE/ActiveX objects before any scripting engine gets invoked with program code extracted from script elements or the value of event attributes.

This has the following two important implications:

a) Programs can be written such that they already refer to and address OLE/ActiveX objects brought in via the host application. E.g. MSIE pre-registers objects like "`window`" (for user interaction), "`document`" (the root object of the parsed, rendered and displayed HTML document)[9] and all elements carrying an "`id`" attribute[10].

b) MSIE can employ *any* Windows Script Engine (WSE) adhering to the interfaces the WSH architecture defines. Among other things this means that IBM's Object Rexx language can be used as a scripting language for HTML in all places where JavaScript can be employed!

Microsoft's version of DOM is dubbed "Dynamic HTML" (DHTML) and is documented extensively via its Microsoft Developer Network (cf. [W3DHTML1, W3DHTML2]), which can be accessed via the Internet/WWW. It documents all

---

[9] Among many other things one can use the "`document`" object for using its collection properties - OLE/ActiveX objects themselves - to analyze and query all or specific elements. E.g. the property "`all`" contains all HTML elements represented as OLE/ActiveX objects in document order. In addition this particular collection serves as a directory which returns the appropriate OLE/ActiveX object, if using the value of the '`id`' attribute of that particular element.

One interesting application of the "`document`" object is the ability to use its methods "`write`" and "`writeln`" to insert text into the HTML document.

[10] Such OLE objects are named exactly like their attribute value by Internet Explorer: e.g. a HTML paragraph markup like '`<p id="myElement">`' allows that particular p-element to be immediately referred to as an OLE object by the name "myElement".

available properties together with all its functionality and gives numerous examples on how to put them to work. In addition there are numerous tutorials and documents explaining the concepts and overall functionality made available via DHTML/DOM.

MSIE accepts script elements with program code *not* enclosed in a CDATA-section! Obviously, MSIE uses the end tag "</script>" as a marker and does not parse the text before it as HTML text.[11)]

3.1 Creating Text with Object Rexx

Figure 7 depicts text marked up with HTML with a script element in the head element and one in the body element. The resulting rendering of that text by MSIE is shown in figure 8.

As can be seen the OLE/ActiveX object named "`document`" is available right away to Object Rexx, as MSIE has pre-registered that object before the code of the script element gets presented to Object Rexx for execution. The text - including the markup - for the title element as well as for the content of the entire body element is created by the Object Rexx code of the script elements.

Referring to the pre-registered object "document" is straight forward as Object Rexx will use the pre-registered OLE/ActiveX object set via the WSH interface by MSIE.

```
<html>
  <head>
     <script language="Object Rexx">
        document~writeln("<title>Produced by Rexx # 1</title>")
     </script>
  </head>
  <body>
     <script language="Object Rexx">
        document~writeln("It is:<em>" date("s") time()"</em>, isn't it?")
     </script>
  </body>
</html>
```

Figure 7: HTML text with Object Rexx program code.

Interaction with these objects occurs as if they were native Object Rexx object, as

[11)] With other words: the characters bearing a special meaning to HTML parsers ("<", ">" and "&") are ignored within the text, enclosed by the "script" tag.

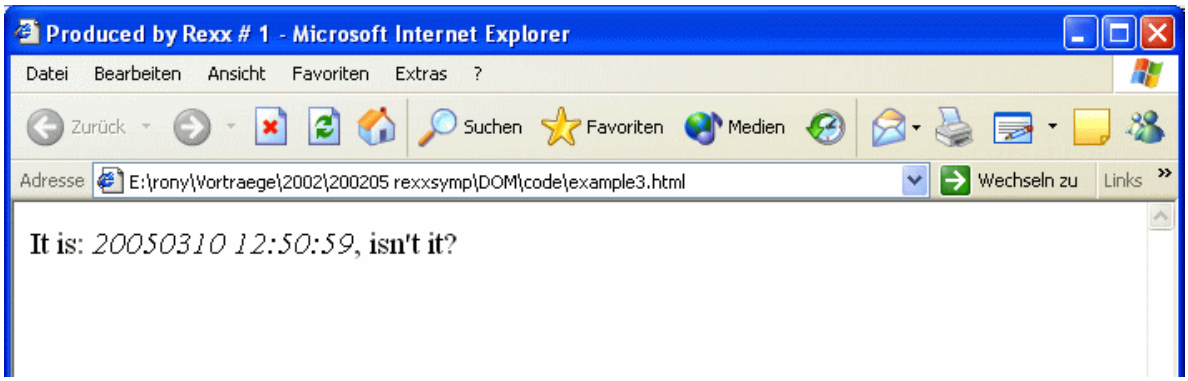

Figure 8: Possible rendering of the HTML text in figure 7, if run on 2005-03-10 12:50:59.2

Object Rexx uses the proxy class OLEObject[12)] to create the Object Rexx representations for those OLE/ActiveX objects. Hence, it is possible to send those proxy objects Object Rexx messages which get transformed to the appropriate OLE/ActiveX invocations on the Windows side.

3.2 Interacting with the Parse Tree

Figure 9 depicts text marked up with HTML with a script element in the head element and one in the body element. In addition to the code in figure 7 there is also a public Object Rexx routine defined, which gets called if a click event occurs in the displayed document. The resulting rendering of that text by MSIE is shown in figure 10.

The body element contains an attribute named "`onclick`" and defines as its value that the public Object Rexx routine "`info`" is to be called with the WSH pre-set OLE/ActiveX object named "`this`" as an argument, which is set to represent exactly the element for which the event was intercepted.[13)]

The public routine "`info`" in turn first queries the received object for its tagname (the method "tagname" gets this particular information from the OLE object) and

---

12) Instances of this proxy class are called "proxy objects", representing with the means of Object Rexx in effect OLE/ActiveX objects.

13) As no embedded element re-defines the "`onclick`" attribute eventually the body element will receive this event and "fire", i.e. react upon receipt of that particular event as defined by calling the public Object Rexx routine "`info`".

```
<html>
  <head>
    <script language="Object Rexx">

      document~writeln("<title>Produced by Rexx # 2</title>")

      ::routine info public -- to be called later on
        use arg o

        output_area~innertext=""
        tmp1="this=["o~tagName"] innerText=["o~innerText"]" "0d0a0d0a"x || ,
             "this=["o~tagName"] innerHtml=["o~innerHtml"]" "0d0a0d0a"x || ,
             "this=["o~tagName"] outerHtml=["o~outerHtml"]" "0d0a0d0a"x

        tmpStr=""
        do item over document~all  -- iterate over all elements
           tmpStr=tmpStr item~tagName
        end
        tmp1=tmp1 || "elements:" tmpStr
        output_area~innertext=tmp1

    </script>
  </head>

  <body onclick="call info this" language="Object Rexx">

     <script language="Object Rexx">
        document~writeln("It is:<em>" date("s") time()"</em>, isn't it?")
     </script>

       <p id="output_area">
  </body>
</html>
```

Figure 9: HTML text containing a public Object Rexx routine "`test`", which gets invoked, whenever an "`onclick`" event occurs.

then for the text it represents (method "innertext"), which is the result of carrying out whatever was defined in the HTML markup and shown in figure 10. The next information queried from it relates to the HTML marked up text containing the script element which was used to produce the displayed text (method "innerhtml"). Finally, the object is queried to return the HTML form of the entire element including its own tags.

Then, the routine will iterate over the OLE objects contained in the collection named "`all`", representing all HTML elements as OLE objects in document order. Hence, the loop will create a list of the HTML tags as they appear in the source document.

Lastly the created string stored with the Object Rexx variable named "`tmp1`" is used to set the text for the element with the "`id`" attribute of "`output_area`", which initially is empty[14]. This in effect replaces whatever text that particular node displays with the string contained in the Rexx variable. This change of a node is reflected right away by the Internet Explorer, changing the content of the WWW browser window right away. Its result can be seen in figure 12 shows the three values shown.

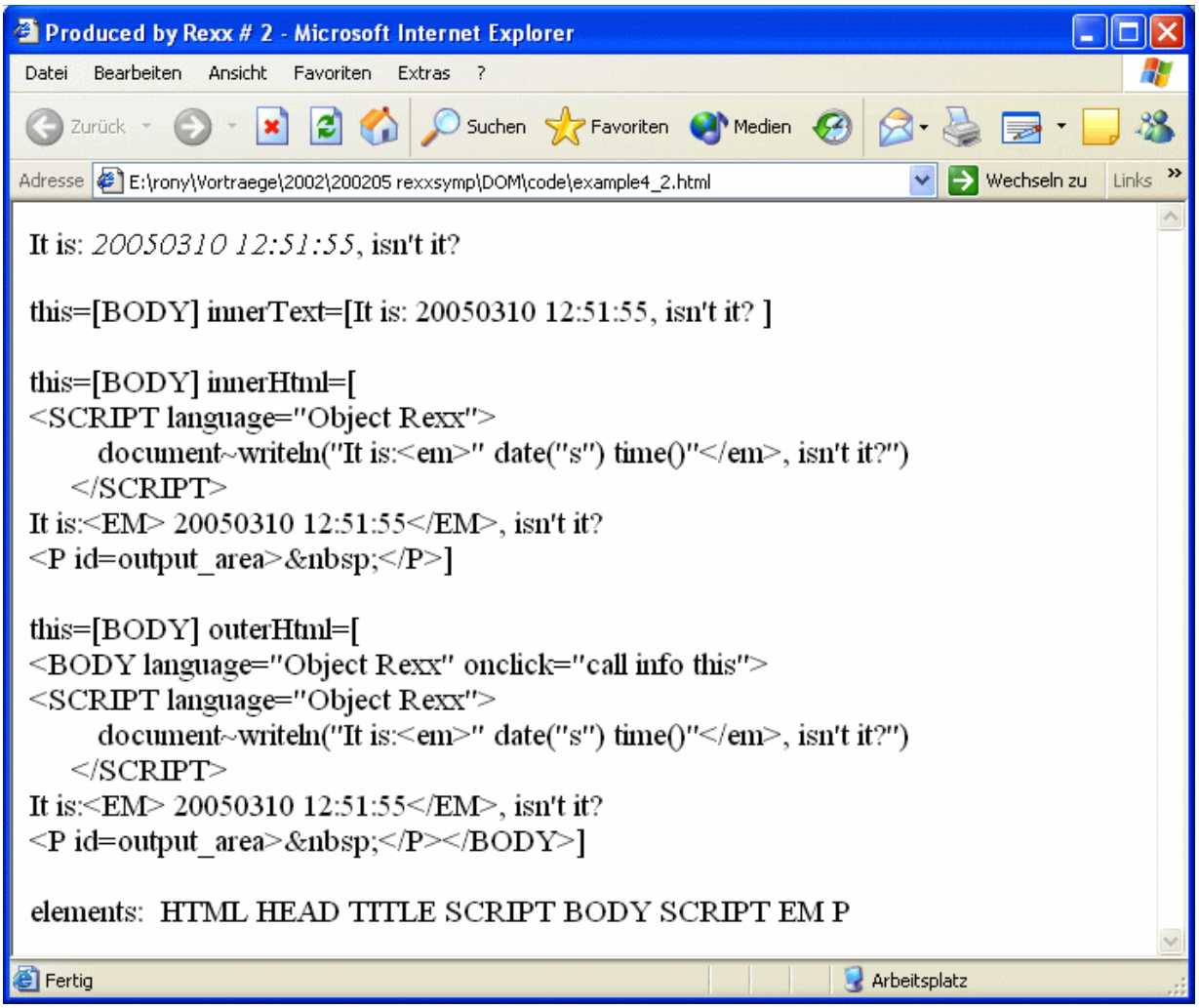

Figure 10: Possible rendering of the HTML text in figure 9, if run on 2005-03-10 12:51:55, after clicking with the mouse anywhere in the browser window.

[14] As can be seen from figure 10, Internet Explorer may add an ending tag for the paragraph tag "P" enclosing the entity named "` `" which is defined to be a "non-breaking space", hence displaying a (non-visible) space.

3.3 Using the Browser as a Graphical User Interface to Communicate with the user

Combining the abilities made available by DOM (and hence DHTML) one can use events for reacting upon user actions, like pressing of a key or button, clicking of a mouse button and the like. As a reaction to user events one can use DOM to change the content of a node or the overall structure of the parse tree of the displayed HTML document.

Figure 11 depicts a HTML page in which HTML "`input`" elements are defined for allowing a user a) to enter text into a textfield (this HTML input element has the attribute "`id`" set to the value "`inputElement`") and b) to press a push button (this HTML input element has the attribute "`id`" set to the value "`outputArea`").

Pressing the push button causes the "`onclick`" event to be intercepted with that element carrying out the associated Object Rexx statement (calling the public Object Rexx routine named "`doTheWork`"). This routine will read the present value of the text area HTML input element ("`inputElement`"), reverse with the appropriate

```
<head>
    <title>Object Rexx: Processing User Input</title>

    <script language="Object Rexx">
        -- queries value of user input, reverses the text and displays it
        ::routine doTheWork public
          outputArea~innerhtml=reverse(inputElement~value)
    </script>

</head>

<body>
    Please enter some text:<p>

    <input id="inputElement" type="textarea" size="80">
    <p> <!-- new paragraph (larger line break) -->

    <input type="BUTTON"
           onclick="call doTheWork" language="Object Rexx"
           value="Please press me!">
    <p> <!-- new paragraph (larger line break) -->

    <p id="outputArea">
</body>
```

Figure 11: HTML text containing Object Rexx code for interacting with the user.

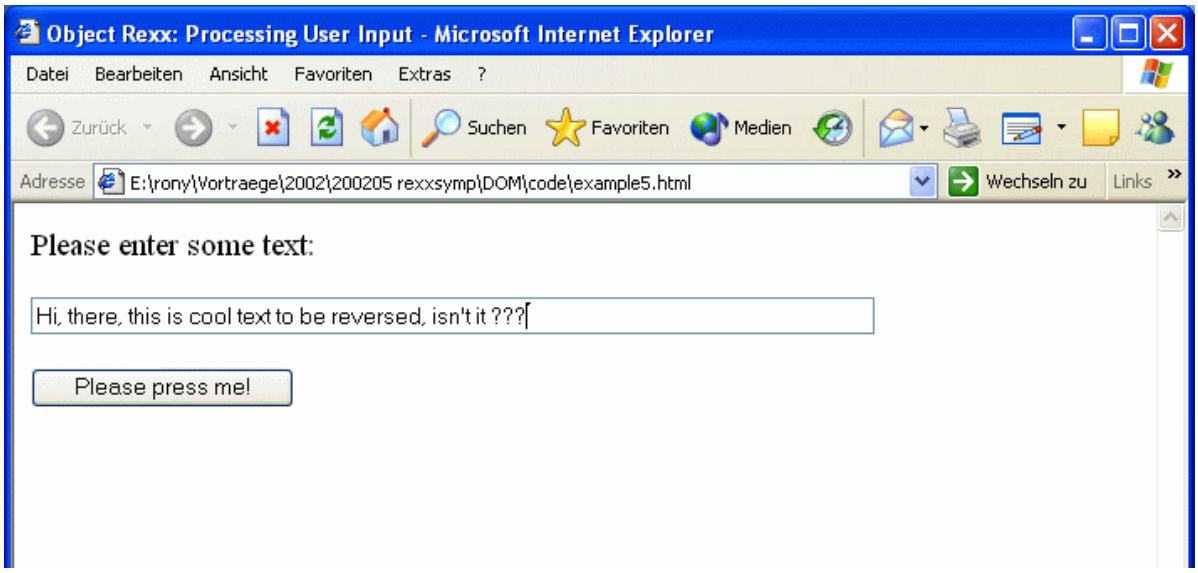

Figure 12: Rendering of the HTML text in figure 11 and text entered by the user.

Rexx built-in function the string and put the result into the paragraph of the HTML element ("`outputArea`").

Figure 12 displays that HTML file and shows the result of the user entering the text "`Hi, there, this is cool text to be reversed, isn't it ???`". Pressing the push button will then yield the rendering of figure 13, displaying the reversed text: "`??? ti t'nsi ,desrever eb ot txet looc si siht ,ereht ,iH`".

Figure 13: Rendering of figure 12, after pressing the push button.

3.4 Some Security Considerations and "HTML-Applications"

HTML (and XML) files are usually loaded from the public Internet. Therefore it is very important that programs embedded in such documents cannot harm the computer on which they run. For this reason MSIE possesses the ability to define security zones which determine the rights of such HTML/XML embedded programs to access the resources of the host computer.

The Object Rexx implementation uses its built-in Security Manager for implementing the security policy for Object Rexx programs according to the settings of MSIE.

For applications using the MSIE as a GUI platform and printing processor it may be desirable to allow unrestricted access to local resources. Such a need may arise for in-house developed applications, where the authors are known and trusted. Microsoft supplies with the Internet Explorer a component which allows to treat such "HTML applications" as regular applications. One merely needs to rename such HTML-files to the extension "`.hta`", the acronym for "HTML application".

It is possible to define some additional characteristics for HTA-files, like an identification string or whether such an application can be invoked just once (just one copy allowed to run) and some more. This information needs to be supplied by additional mark-up documented in the Microsoft developer network.

4 SUMMARY AND OUTLOOK

This article attempted to give an introduction to the basic concepts of DOM and the Microsoft implementation of it: DHTML. In order to understand the principles of DOM, it is important to know the basic rules of markup languages like HTML and also accompanying technologies like "Cascading Style Sheets" (CSS) adding enormous formatting power.

After learning about the hierarchical structure of HTML (and XML markup as well) examples of HTML files employing Object Rexx scripts are introduced and explained. These Object Rexx programs work with Microsoft's Internet Explorer (MSIE) only, because they take advantage of the Microsoft Script technology of which MSIE is a Windows script host application and the Windows version of Object Rexx is a Windows script engine.

They demonstrate the principles of employing the Microsoft implementation of DOM, called DHTML (dynamic HTML) and add from example to example more key features. The last example even demonstrates the possibility to use HTML pages to define graphical user interfaces (GUI) and add programming logic implemented with Object Rexx to it.

One very important conclusion of this article is the following: given some knowledge of HTML, CSS and DHTML, it becomes feasible for any Object Rexx programmer to take advantage of this technology in two distinct benefitiary manner: a) using the Internet Explorer as an easy to use GUI interface technology for Object Rexx programmed frontends and b) using the Internet Explorer as a printing engine, capable of printing the most demanding formats to any of the Windows supplied printers[15)].

[15)] The author is confident that programmers having created programs that interface directly with complex printers will more than appreciate this aspect!

5 REFERENCES

[W3C] Homepage of the World Wide Web Consortium, URL (2002-06-15): http://www.w3.org/

[W3CSS] Homepage of the World Wide Web Consortium project "Cascading Style Sheets (CSS)", URL (2002-06-15): http://www.w3.org/CSS

[W3DOM] Homepage of the World Wide Web Consortium project "Document Object Model (DOM)", URL (2002-06-15): http://www.w3.org/DOM

[W3HTML] Homepage of the World Wide Web Consortium project "Hypertext Markup Language (HTML)", URL (2002-06-15): http://www.w3.org/Markup

[W3DHTML1] Homepage of the Microsoft Developer Network for "Dynamic Hypertext Markup Language (DHTML)", URL (2002-06-15): http://msdn.microsoft.com/workshop/author/dhtml/dhtml_node_entry.asp

[W3DHTML2] Homepage of the Microsoft Developer Network for "DHTML Reference", URL (2002-06-15): http://msdn.microsoft.com/workshop/author/dhtml/reference/dhtml_reference_entry.asp

[W3ORX] Object Rexx homepage of IBM, URL (2002-06-15): http://www.ibm.com/software/ad/obj-rexx/

[W3WSH] Microsoft's scripting homepage, URL (2002-06-15): http://msdn.microsoft.com/scripting

[W3XML] Homepage of the World Wide Web Consortium project "Extensible Markup Language (XML)", URL (2002-06-15): http://www.w3.org/XML

Date of Article: 2002-06-15.

Published in: Proceedings of the „2002 International Rexx Symposium“, April 29th - May 1th, Raleigh, NC, USA 2002.

Presented at: „The 2002 International Rexx Symposium“, April 29th - May 1th, Raleigh, NC, USA 2002.