MIS Department

Abteilung für Wirtschaftsinformatik

Processing XML Documents with DOM Using BSF4ooRexx

2013 International Rexx Symposium

RTP, North Carolina

Prof. Dr. Rony G. Flatscher

Vienna University of Economics and Business Administration ■ Wirtschaftsuniversität Wien ■ Augasse 2-6 ■ A-1090 Wien

© 2013 Rony G. Flatscher

Overview

- Markup-Language
  - Basics
  - XML
- DOM
  - Principles
  - Exploiting DOM from ooRexx
  - Examples
- Roundup

© 2013 Rony G. Flatscher

Terms, 1
(Markup Languages)

- Tag
  - Enables one to use tags to embrace regular text
    - Opening tag (a.k.a. start tag)

      `<some_tag_name>`
    - Closing tag (a.k.a. end tag)

      `</some_tag_name>`
  - Allows for analyzing text, by noticing which parts of a text are surrounded ("embraced") by which tags
  - "Element"
    - The sequence "opening tag", text, "closing tag"

© 2013 Rony G. Flatscher

Terms, 2
(Markup Languages)

- Document Type Definition (DTD)
  - Defines the tags and their attributes, if any
    - Name (identifier) of the tag
    - Attributes for tags
    - "Content model"
      - Nesting of tags and the allowed sequence of tags
        - **Hierarchical structure !**
      - Allows to determine how many times an element may occur
  - "Instance" of a DTD
    - A document with text that got marked-up according to the rules defined in a DTD
    - A document that has been checked whether the DTD rules were applied correctly is named a "valid" document

© 2013 Rony G. Flatscher

Terms, 3 (Markup Languages)

- HTML
  - A markup language for the WWW
    - HTML-Browser
      - Parses a document marked up according to HTML
      - Formats the text, depending on the used tags
  - DTD
    - Version 4.01: three variants defined
    - SGML-based, hence it is possible to
      - Use any case for the tags and attribute names
      - Some closing tags may be omitted if the end tags can be determined by the rules set forth in the DTD
      - It is possible to define exclusions

© 2013 Rony G. Flatscher

Terms, 4 (Markup Languages)

- XML
  - A slightly simplified version of SGML
    - Allows the definition of DTDs for markup languages
      - Since 2002 an alternative got introduced in the form of "XML Schema": `http://www.w3c.org`
    - Tag and attribute names must be written in exact case
    - End tags must be always given
    - Attribute values can now be enclosed within apostrophes/single quotes (') in addition to double quotes (")
    - It is possible to explicitly denote empty elements

```
<some_tag_name/>
```


© 2013 Rony G. Flatscher

Terms, 5 (Markup Languages)

- XML DTDs can be omitted
  - A matching DTD can be always inferred, if the document is "well formed":
    - All tags must be nested
    - Tags must not overlap
    - Start tags must have matching end tags
- Structure is always independent of the formatting!
  - Cascading Style Sheets (CSS)
    - Allows to define formatting (layout) rules for elements
    - It is possible to define specific formatting (layout) rules for elements with attributes that have specific values or depending on the sequence of the elements

© 2013 Rony G. Flatscher

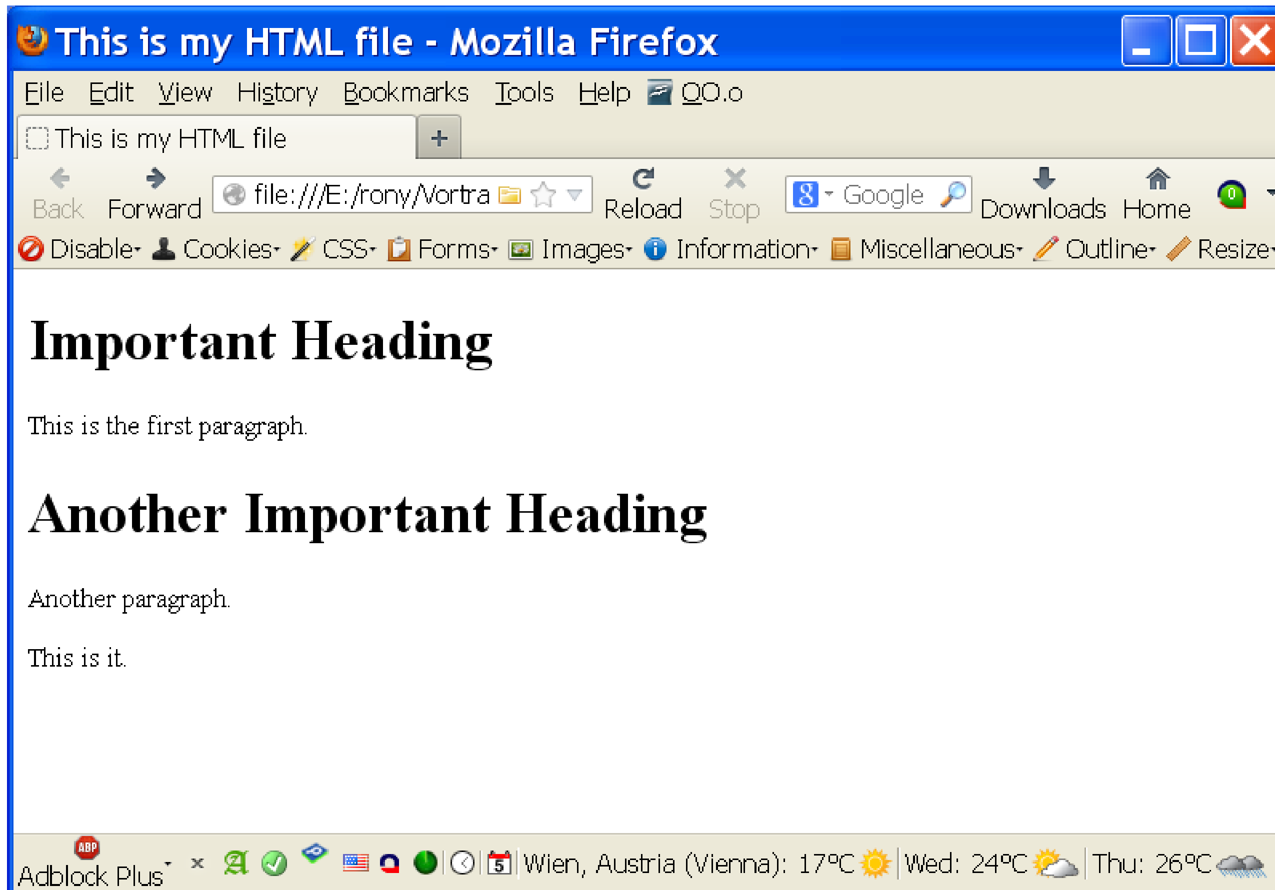
Terms (XHTML)

- Text, marked up in XHTML

```
<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.</p>
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.</p>
     <p id="a9876">This <span class="verb">is</span> it.</p>
  </body>
</html>
```


© 2013 Rony G. Flatscher

XHTML Text Rendered in Firefox



© 2013 Rony G. Flatscher

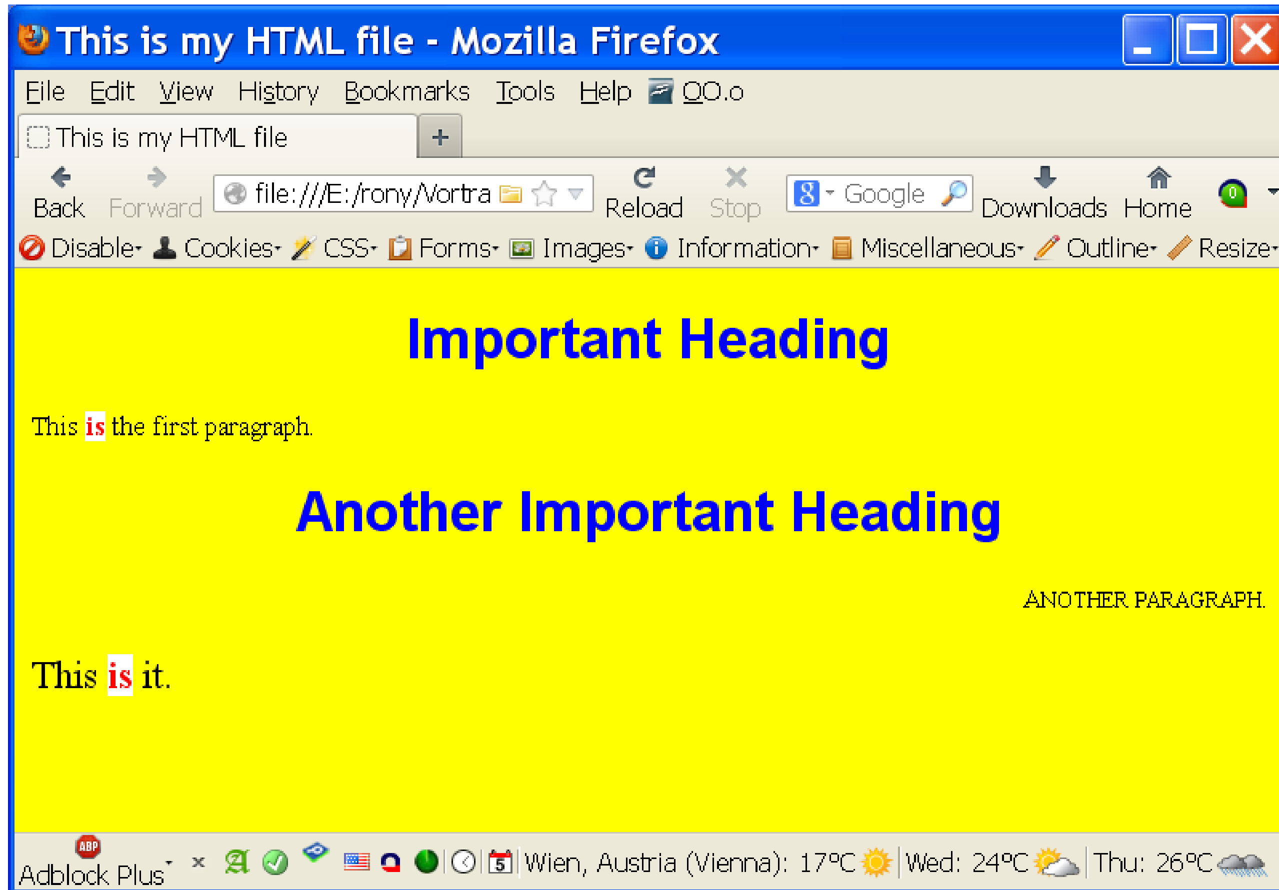
Example: Linking a Cascading Style Sheet (CSS)

- Text, marked up in XHTML

```
<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.</p>
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.</p>
     <p id="a9876">This <span class="verb">is</span> it.</p>
  </body>
</html>
```


© 2013 Rony G. Flatscher

XHTML Text Rendered in Firefox with CSS



© 2013 Rony G. Flatscher

Example of a Cascading Style Sheets (CSS)

Tag → h1
Tag → body
"class" Attribut → .verb
"id" Attribut → #xyz1
"id" Attribut → #a9876

```
h1        { color: blue;
            text-align: center;
            font-family: Arial,sans-serif;
            font-size: 200%; }

body      { background-color: yellow;
            font-family: Times, Avantgarde;
            font-size: small; }

.verb     { background-color: white;
            color: red;
            font-weight: 900; }

#xyz1     { font-variant: small-caps;
            text-align: right; }

#a9876    { font-size: large; }
```


© 2013 Rony G. Flatscher

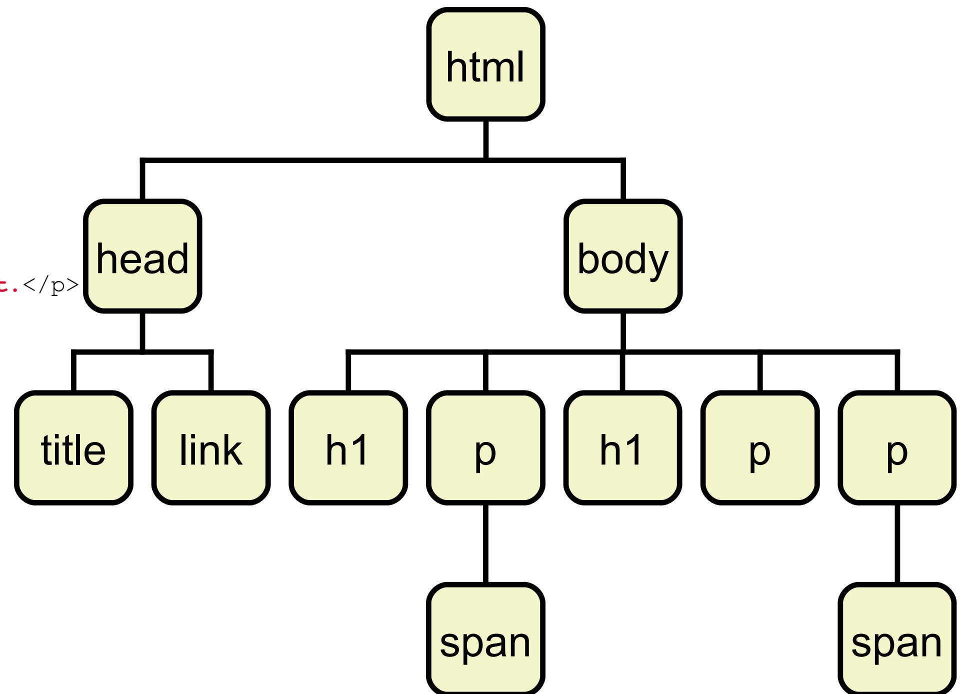
Document Object Model (DOM), 1

- Parsing of a HTML/XML file
  - Creates a parse tree in which the elements are nodes
    - Each HTML browser needs to do this with each HTML file!
- Application Programming Interfaces (API) for
  - Creating, querying, changing and deleting nodes from the tree
    - This includes the manipulation of attributes as well!
    - Intercepting events, while the user is working with the parse tree
    - User generated events as a result of using the mouse or the keyboard
    - Application generated events like "document loaded"

© 2013 Rony G. Flatscher

Document Object Model (DOM), 2 Example

```
<!-- example2.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml1-transitional.dtd">
<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.</p>
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.</p>
     <p id="a9876">This <span class="verb">is</span> it.</p>
  </body>
</html>
```



© 2013 Rony G. Flatscher

DOM (Document Object Model), 3
Java DOM Parsers

- A Java DOM parser processes the entire document
  - "Factory" pattern, which allows different implementations of DOM parsers to be deployed via Java
- The DOM parser creates a parse tree where each node is one of type (cf. Java documentation of org.w3c.dom.Node)
  - Attr (attribute), CDATASection (character data section), Comment, Document, DocumentFragment, DocumentType, Element, Entity, EntityReference, Notation, ProcessingInstruction, Text
- The interface definitions for nodes (cf. org.w3c.dom.Node) also include a set of methods to manipulate the parse tree

© 2013 Rony G. Flatscher

DOM (Document Object Model), 4

- The interface org.xml.sax.ErrorHandler defines the methods a SAX/DOM error listener must implement
  - error(SAXParseException exception)
  - fatalError(SAXParseException exception)
  - warning(SAXParseException exception)
- org.xml.sax.SAXParseException has the following methods
  - getCause() returns a Throwable Java object representing the cause
  - getException() returns an embedded exception, if any
  - getMessage() returns a string with the detailed error message
  - toString() returns a string representation of the SAXParseException

© 2013 Rony G. Flatscher

DOM-parsing in ooRexx

- Create an ooRexx listener class for handling errors/warnings
- Create an ooRexx listener object from it
- Create a Java object that embeds the ooRexx listener object
  - BSFCreateRexxProxy(rexxListenerObject,[slotArg],interfaceName[,...])
  - interfaceName denotes the Java interface name which methods the Rexx listener object handles
    - It is possible to denote more than one Java interface, if the Rexx listener object is able to handle all methods defined by them!
- Let the Java DOM parser parse the document
- Process the resulting parse tree with Rexx routines node by node

© 2013 Rony G. Flatscher

"code01.rxj ": Extract Text from any XHTML Document The ooRexx Program (Main Program), 1

```
/* purpose: demonstrate how to extract the text from a xml file using DOM */
parse arg xmlFileName

   /* create an instance of the JAXP DocumentBuilderFactory    */
factory=bsf.loadClass("javax.xml.parsers.DocumentBuilderFactory")~newInstance
factory~setNamespaceAware(.true)     -- set desired parser to namespace aware
parser=factory~newDocumentBuilder    -- create the parser from the factory

eh=.errorHandler~new        -- create an error handler Rexx object
   -- wrap up the Rexx error handler as a Java object
javaEH=BsfCreateRexxProxy(eh, , "org.xml.sax.ErrorHandler")
parser~setErrorHandler(javaEH)    -- set the error handler for this parser

rootNode=parser~parse(xmlFileName)  -- parse the file, returns root node

   /* make important constants available via .local            */
clzDomNode=bsf.loadClass("org.w3c.dom.Node") -- load the Java interface class
.local~CDATA_SECTION_NODE=clzDomNode~CDATA_SECTION_NODE  -- save field value
.local~TEXT_NODE         =clzDomNode~TEXT_NODE           -- save field value

   /* now collect all text and CDATA nodes and display them    */
call followNode rootNode

::requires BSF.CLS      /* get the Java support */

... cut ...
```


© 2013 Rony G. Flatscher

"code01.rxj ": Extract Text from any XHTML Document The ooRexx Program (the ooRexx Classes), 2

```
... cut ...
::requires BSF.CLS        /* get the Java support */

::routine followNode     /* walks the document tree recursively */
  use arg node
  call processNode node               -- process received node
  if node~hasChildNodes then
  do
     children=node~getChildNodes      -- get NodeList
     loop i=0 to children~length-1    -- 0-based indexes!
        call followNode children~item(i)   -- recurse
     end
  end

::routine processNode    /* processes each node                 */
  use arg node
  nodeType=node~getNodeType       -- get type of node
  if nodeType=.text_node | nodeType=.cdata_section_node then
     say pp(node~nodeValue)

::class ErrorHandler     -- a Rexx error handler ("org.xml.sax.ErrorHandler")

::method unknown         /* handles "warning", "error" and "fatalError" events */
  use arg methName, argArray  -- arguments from the Java SAX parser
  exception=argArray[1] -- retrieve SAXException argument
  .error~say(methName":" -
            "line="exception~getLineNumber",col="exception~getColumnNumber":" -
            pp(exception~getMessage))
```


© 2013 Rony G. Flatscher

"code01.rxj ": Extract Text from any XHTML Document Running the ooRexx Program, 3

```
f:\>rexx code01_text.rxj example2.html
[
  ]
[
      ]
[This is my HTML file]
[
      ]
[
  ]
[
  ]
[
      ]
[Important Heading]
[
      ]
[This ]
[is]
[ the
         first paragraph.]
[
      ]
[Another Important Heading]
[
      ]
[Another paragraph.]
[
      ]
[This ]
[is]
[ it.]
[
  ]
[
]
```

```
<!-- example2.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "DTD/xhtml1-transitional.dtd">
<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.</p>
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.</p>
     <p id="a9876">This <span class="verb">is</span> it.</p>
  </body>
</html>
```


© 2013 Rony G. Flatscher

"code01.rxj ": Extract Text from any XHTML Document Some Remarks, 4

- Some remarks
  - Text can be encoded as
    - Plain text (node type TEXT_NODE) or
    - CDATA sections (node type CDATA_SECTION_NODE)
      - **<[!CDATA[**...character-data...**]]>**
  - Ignorable whitespace is not ignored, but treated like any whitespace
    - A node of type TEXT_NODE will be created for it
  - The node types used in the Java DOM parser are retrievable via the Java interface class org.w3c.dom.Node
    - To make it easy to refer to these values from ooRexx, the types TEXT_NODE and CDATA_SECTION_NODE are retrieved and made available to all parts of the Rexx program by storing them in .local

© 2013 Rony G. Flatscher

"code02.rxj": List Elements in Document Order The ooRexx Program (Main Program), 1

```
/* purpose: demonstrate how to extract the text from a xml file using DOM */
parse arg xmlFileName

   /* create an instance of the JAXP DocumentBuilderFactory    */
factory=bsf.loadClass("javax.xml.parsers.DocumentBuilderFactory")~newInstance
factory~setNamespaceAware(.true)     -- set desired parser to namespace aware
parser=factory~newDocumentBuilder    -- create the parser from the factory

eh=.errorHandler~new        -- create an error handler Rexx object
   -- wrap up the Rexx error handler as a Java object
javaEH=BsfCreateRexxProxy(eh, , "org.xml.sax.ErrorHandler")
parser~setErrorHandler(javaEH)    -- set the error handler for this parser

rootNode=parser~parse(xmlFileName)  -- parse the file, returns root node

   /* make important constants available via .local            */
clzDomNode=bsf.loadClass("org.w3c.dom.Node") -- load the Java interface class
.local~ELEMENT_NODE      =clzDomNode~ELEMENT_NODE   -- save field value

   /* now collect all text and CDATA nodes and display them    */
call followNode rootNode

::requires BSF.CLS      /* get the Java support */

... cut ...
```


© 2013 Rony G. Flatscher

"code02.rxj": List Elements in Document Order The ooRexx Program (the ooRexx Classes), 2

```
... cut ...
::requires BSF.CLS      /* get the Java support */

::routine followNode    /* walks the document tree recursively */
  use arg node
  call processNode node               -- process received node
  if node~hasChildNodes then
  do
     children=node~getChildNodes      -- get NodeList
     loop i=0 to children~length-1    -- 0-based indexes!
        call followNode children~item(i)   -- recurse
     end
  end

::routine processNode   /* processes each node                  */
  use arg node
  nodeType=node~getNodeType        -- get type of node
  if nodeType=.element_node then
     say pp(node~nodeName)

::class ErrorHandler    -- a Rexx error handler ("org.xml.sax.ErrorHandler")

::method unknown        /* handles "warning", "error" and "fatalError" events */
  use arg methName, argArray  -- arguments from the Java SAX parser
  exception=argArray[1] -- retrieve SAXException argument
  .error~say(methName":" -
           "line="exception~getLineNumber",col="exception~getColumnNumber":" -
           pp(exception~getMessage))
```


© 2013 Rony G. Flatscher

"code02.rxj": List Elements in Document Order Running the ooRexx Program, 3

```
f:\>rexx code02_text.rxj example2.html
[html]
[head]
[title]
[link]
[body]
[h1]
[p]
[span]
[h1]
[p]
[p]
[span]
```

```
<!-- example2.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "DTD/xhtml1-transitional.dtd">
<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.</p>
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.</p>
     <p id="a9876">This <span class="verb">is</span> it.</p>
  </body>
</html>
```


© 2013 Rony G. Flatscher

"code02.rxj": List Elements in Document Order Some Remarks, 4

- A few remarks
  - The DOM parse tree has more nodes than shown in the output!
  - This particular program processes nodes of type ELEMENT_NODE only
  - The node types used in the Java DOM parser are retrievable via the Java interface class org.w3c.dom.Node
    - To make it easy to refer to these values from ooRexx, the type ELEMENT_NODE is retrieved and made available to all parts of the Rexx program by storing it in .local
- One could also use the Java infrastructure to filter only those node types one is interested in

© 2013 Rony G. Flatscher

"code03.rxj" : List Elements in Document Order #2 The ooRexx Program (Main Program), 1

```
/* purpose: demonstrate how to extract the text from a xml file using DOM */
parse arg xmlFileName

   /* create an instance of the JAXP DocumentBuilderFactory    */
factory=bsf.loadClass("javax.xml.parsers.DocumentBuilderFactory")~newInstance
factory~setNamespaceAware(.true)     -- set desired parser to namespace aware
parser=factory~newDocumentBuilder    -- create the parser from the factory

eh=.errorHandler~new        -- create an error handler Rexx object
   -- wrap up the Rexx error handler as a Java object
javaEH=BsfCreateRexxProxy(eh, , "org.xml.sax.ErrorHandler")
parser~setErrorHandler(javaEH)    -- set the error handler for this parser

rootNode=parser~parse(xmlFileName)  -- parse the file, returns root node

   /* make important constants available via .local            */
clzDomNode=bsf.loadClass("org.w3c.dom.Node") -- load the Java interface class
.local~ELEMENT_NODE      =clzDomNode~ELEMENT_NODE  -- save field value

   /* now collect all text and CDATA nodes and display them    */
call followNode rootNode, 0

::requires BSF.CLS      /* get the Java support */

... cut ...
```


© 2013 Rony G. Flatscher

"code03.rxj" : List Elements in Document Order #2 The ooRexx Program (the ooRexx Classes), 2

```
... cut ...
::requires BSF.CLS       /* get the Java support */

::routine followNode     /* walks the document tree recursively */
  use arg node, level
  call processNode node, level        -- process received node
  if node~hasChildNodes then
  do
     children=node~getChildNodes     -- get NodeList
     loop i=0 to children~length-1   -- 0-based indexes!
        call followNode children~item(i), level+1  -- recurse
     end
  end

::routine processNode    /* processes each node                 */
  use arg node, level
  nodeType=node~getNodeType       -- get type of node
  if nodeType=.element_Node then
     say "    "~copies(level) || pp(node~nodeName)

::class ErrorHandler     -- a Rexx error handler ("org.xml.sax.ErrorHandler")

::method unknown         /* handles "warning", "error" and "fatalError" events */
  use arg methName, argArray  -- arguments from the Java SAX parser
  exception=argArray[1] -- retrieve SAXException argument
  .error~say(methName":" -
           "line="exception~getLineNumber",col="exception~getColumnNumber":" -
           pp(exception~getMessage))
```


© 2013 Rony G. Flatscher

"code03.rxj" : List Elements in Document Order #2 Running the ooRexx Program, 3

```
f:\>rexx code03_text.rxj example2.html
    [html]
        [head]
            [title]
            [link]
        [body]
            [h1]
            [p]
                [span]
            [h1]
            [p]
            [p]
                [span]
```

```
<!-- example2.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "DTD/xhtml1-transitional.dtd">
<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.</p>
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.</p>
     <p id="a9876">This <span class="verb">is</span> it.</p>
  </body>
</html>
```


© 2013 Rony G. Flatscher

"code03.rxj" : List Elements in Document Order #2 Some Remarks, 4

- A few remarks
  - The DOM parse tree has more nodes than shown in the output!
    - Note whitespace before first shown element
  - This particular program processes nodes of type ELEMENT_NODE only
  - The node types used in the Java DOM parser are retrievable via the Java interface class org.w3c.dom.Node
    - To make it easy to refer to these values from ooRexx, the type ELEMENT_NODE is retrieved and made available to all parts of the Rexx program by storing it in .local
- One could also use the Java infrastructure to filter only those node types one is interested in

© 2013 Rony G. Flatscher

"code04.rxj" : List Elements with Text
The ooRexx Program (Main Program), 1

```
/* purpose: demonstrate how to extract the text from a xml file using DOM */
parse arg xmlFileName

   /* create an instance of the JAXP DocumentBuilderFactory    */
factory=bsf.loadClass("javax.xml.parsers.DocumentBuilderFactory")~newInstance
factory~setNamespaceAware(.true)     -- set desired parser to namespace aware
parser=factory~newDocumentBuilder    -- create the parser from the factory

eh=.errorHandler~new       -- create an error handler Rexx object
   -- wrap up the Rexx error handler as a Java object
javaEH=BsfCreateRexxProxy(eh, , "org.xml.sax.ErrorHandler")
parser~setErrorHandler(javaEH)    -- set the error handler for this parser

rootNode=parser~parse(xmlFileName)  -- parse the file, returns root node

   /* make important constants available via .local            */
clzDomNode=bsf.loadClass("org.w3c.dom.Node") -- load the Java interface class
.local~CDATA_SECTION_NODE=clzDomNode~CDATA_SECTION_NODE  -- save field value
.local~TEXT_NODE         =clzDomNode~TEXT_NODE           -- save field value
.local~ELEMENT_NODE      =clzDomNode~ELEMENT_NODE   -- save field value

   /* now collect all text and CDATA nodes and display them    */
call followNode rootNode, 0

::requires BSF.CLS      /* get the Java support */

... cut ...
```


© 2013 Rony G. Flatscher

"code04.rxj" : List Elements with Text
The ooRexx Program (the ooRexx Classes), 2

```
... cut ...
::requires BSF.CLS       /* get the Java support */

::routine followNode     /* walks the document tree recursively */
  use arg node, level
  call processNode node, level       -- process received node
  if node~hasChildNodes then
  do
     children=node~getChildNodes     -- get NodeList
     loop i=0 to children~length-1   -- 0-based indexes!
        call followNode children~item(i), level+1   -- recurse
     end
  end

::routine processNode    /* processes each node                  */
  use arg node, level
  nodeType=node~getNodeType       -- get type of node
  if nodeType=.text_node | nodeType=.cdata_section_node then
     say "   "~copies(level) || "-->" pp(node~getNodeValue)    -- instead of getData()
  else if nodeType=.element_node then
     say "   "~copies(level) || pp(node~getNodeName)

::class ErrorHandler     -- a Rexx error handler ("org.xml.sax.ErrorHandler")

::method unknown         /* handles "warning", "error" and "fatalError" events */
  use arg methName, argArray  -- arguments from the Java SAX parser
  exception=argArray[1] -- retrieve SAXException argument
  .error~say(methName":" -
           "line="exception~getLineNumber",col="exception~getColumnNumber":" -
           pp(exception~getMessage))
```


© 2013 Rony G. Flatscher

"code04.rxj" : List Elements with Text Running the ooRexx Program, 3

```
f:\>rexx code04_text.rxj example2.html
 [html]
      --> [
  ]
      [head]
         --> [
     ]
         [title]
            --> [This is my HTML file]
         --> [
     ]
         [link]
         --> [
  ]
      --> [
  ]
      [body]
         --> [
     ]
         [h1]
            --> [Important Heading]
         --> [
     ]
         [p]
            --> [This ]
            [span]
               --> [is]
            --> [ the
        first paragraph.]
         --> [
     ]
         [h1]
            --> [Another Important Heading]
         --> [
     ]
         [p]
            --> [Another paragraph.]
         --> [
     ]
         [p]
            --> [This ]
            [span]
               --> [is]
            --> [ it.]
         --> [
  ]
      --> [
]
```

```
<!-- example2.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "DTD/xhtml1-transitional.dtd">
<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.</p>
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.</p>
     <p id="a9876">This <span class="verb">is</span> it.</p>
  </body>
</html>
```


© 2013 Rony G. Flatscher

"code04.rxj" : List Elements with Text Some Remarks, 4

- A few remarks
  - The DOM parse tree has more nodes than shown in the output!
    - Note whitespace before first shown element
  - This particular program processes nodes of type TEXT_NODE, CDATA_SECTION_NODE and ELEMENT_NODE only
  - The node types used in the Java DOM parser are retrievable via the Java interface class org.w3c.dom.Node
    - To make it easy to refer to these values from ooRexx, the types TEXT_NODE, CDATA_SECTION_NODE and ELEMENT_NODE are retrieved and made available to all parts of the Rexx program by storing them in .local
  - One could also use the Java infrastructure to filter only those node types one is interested in

© 2013 Rony G. Flatscher

More Available with Java DOM Parsing Than Documented by Java!

- The default DOM parser coming with Java (Apache's Xerces2) is capable of more than what is documented in Oracle's JavaDocs!
  - The W3C interface org.w3c.dom.traversal.DocumentTraversal, cf. documentation at *http://www.w3.org/2003/01/dom2-javadoc/org/w3c/dom/traversal/DocumentTraversal.html*
    - Method createNodeIterator(...) filters nodes and returns the result as a list with the methods defined in the interface org.w3c.dom.traversal.NodeIterator for its traversal
    - Method createTreeWalker(...) filters nodes and returns the result as a tree with the methods defined in the interface org.w3c.dom.traversal.TreeWalker for its traversal

© 2013 Rony G. Flatscher

Using A NodeIterator to Iterate over Elements

- Get the Java constant field value for showing (filtering) elements using the Java interface class org.w3c.dom.traversal.NodeFilter and the constant field named SHOW_ELEMENT
- Create a NodeIterator from the DOM parse tree and use its methods to iterate over the filtered nodes
- Hint
  - Compare the following code "code05.rxj" with "code02.rxj" above

© 2013 Rony G. Flatscher

"code05.rxj": List Elements in Document Order The ooRexx Program (Main Program), 1

```
/* purpose: demonstrate how to list the element names in document order */
parse arg xmlFileName
   /* create an instance of the JAXP DocumentBuilderFactory    */
factory=bsf.loadClass("javax.xml.parsers.DocumentBuilderFactory")~newInstance
factory~setNamespaceAware(.true)     -- set desired parser to namespace aware
parser=factory~newDocumentBuilder    -- create the parser from the factory

eh=.errorHandler~new        -- create an error handler Rexx object
   -- wrap up the Rexx error handler as a Java object
javaEH=BsfCreateRexxProxy(eh, , "org.xml.sax.ErrorHandler")
parser~setErrorHandler(javaEH)    -- set the error handler for this parser

rootNode=parser~parse(xmlFileName)  -- parse the file, returns root node

   /* get constant value to determine node types to filter     */
whatToShow=bsf.getConstant("org.w3c.dom.traversal.NodeFilter", "SHOW_ELEMENT")
   /* create a NodeIterator with only Element nodes            */
iterator=rootNode~createNodeIterator(rootNode, whatToShow, .nil, .true)
   /* process list of Element nodes                            */
node=iterator~nextNode  /* get first node                      */
loop while node<>.nil
   say pp(node~getNodeName)
   node=iterator~nextNode  /* get next node                    */
end

::requires BSF.CLS       /* get the Java support */
::class ErrorHandler     -- a Rexx error handler ("org.xml.sax.ErrorHandler")
... as depicted above ...
```


© 2013 Rony G. Flatscher

"code05.rxj": List Elements in Document Order Running the ooRexx Program, 2

```
f:\>rexx code05_text.rxj example2.html
[html]
[head]
[title]
[link]
[body]
[h1]
[p]
[span]
[h1]
[p]
[p]
[span]
```

```
<!-- example2.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "DTD/xhtml1-transitional.dtd">
<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.</p>
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.</p>
     <p id="a9876">This <span class="verb">is</span> it.</p>
  </body>
</html>
```


© 2013 Rony G. Flatscher

Using A TreeWalker to Iterate over Elements

- Get the Java constant field value for showing (filtering) elements using the Java interface class org.w3c.dom.traversal.NodeFilter and the constant field named SHOW_ELEMENT
- Create a TreeWalker from the DOM parse tree and use its methods to iterate over the filtered nodes
- Note
  - The createTreeWalker() method will filter element related nodes as well (e.g. text nodes included in an element)
- Hint
  - Compare the following code "code06.rxj" with "code03.rxj" above

© 2013 Rony G. Flatscher

"code06.rxj": List Elements in Document Order The ooRexx Program (Main Program), 1

```
/* purpose: demonstrate how to list the element names in document order */

parse arg xmlFileName

   /* create an instance of the JAXP DocumentBuilderFactory    */
factory=bsf.loadClass("javax.xml.parsers.DocumentBuilderFactory")~newInstance
factory~setNamespaceAware(.true)     -- set desired parser to namespace aware
parser=factory~newDocumentBuilder    -- create the parser from the factory

eh=.errorHandler~new        -- create an error handler Rexx object
   -- wrap up the Rexx error handler as a Java object
javaEH=BsfCreateRexxProxy(eh, , "org.xml.sax.ErrorHandler")
parser~setErrorHandler(javaEH)    -- set the error handler for this parser

rootNode=parser~parse(xmlFileName)  -- parse the file, returns root node

   /* get constant value to determine node types to filter     */
.local~show_element=bsf.getConstant("org.w3c.dom.traversal.NodeFilter", "SHOW_ELEMENT")
whatToShow=.show_element

   /* create a TreeWalker with only Element nodes              */
walker=rootNode~createTreeWalker(rootNode, whatToShow, .nil, .true)

   /* process list of Element nodes                            */
call walkTheTree walker~firstChild, 0

::requires BSF.CLS      /* get the Java support */

... cut here ...
```


© 2013 Rony G. Flatscher

"code06.rxj": List Elements in Document Order The ooRexx Program, 2

```
... cut here ...

   /* process list of Element nodes                              */
call walkTheTree walker~firstChild, 0

::requires BSF.CLS      /* get the Java support */

::routine walkTheTree   /* walk the tree recursively                  */
  use arg node, level

  say "   "~copies(level) || pp(node~getNodeName)  -- show element name indented

  child=node~firstChild        -- depth first
  do while child<>.nil
     -- there may be other node types coming with elements in a TreeWalker
    if child~getNodeType=.show_element then -- make sure only element nodes
       call walkTheTree child, level+1      -- recurse, increase level

    child=child~nextSibling  -- breadth next
  end


::class ErrorHandler    -- a Rexx error handler ("org.xml.sax.ErrorHandler")

... as depicted above ...
```


© 2013 Rony G. Flatscher

"code05.rxj": List Elements in Document Order Running the ooRexx Program, 2

```
f:\>rexx code06_text.rxj example2.html
[html]
    [head]
        [title]
        [link]
    [body]
        [h1]
        [p]
            [span]
        [h1]
        [p]
        [p]
            [span]
```

```
<!-- example2.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "DTD/xhtml1-transitional.dtd">
<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.</p>
     <h1>Another Important Heading</h1>
     <p id="xyz1">Another paragraph.</p>
     <p id="a9876">This <span class="verb">is</span> it.</p>
  </body>
</html>
```


© 2013 Rony G. Flatscher

Roundup

- Parsing any XML encoded document possible
  - Using BSF4ooRexx
  - Exploiting Java's functionality for parsing XML documents
- DOM parsing
  - DOM parser first creates parse tree
  - One may directly walk the DOM parse tree or use the traversal methods to filter the DOM parse tree into a NodeIterator or TreeWalker
- Rather easy, needs enough memory for the parse tree
- Easy to exploit from ooRexx !

© 2013 Rony G. Flatscher

Further Information

- World Wide Web Consortium ("W3C")
  http://www.w3c.org
  http://www.w3c.org/Style/CSS/
  http://www.w3c.org/DOM/
  http://www.w3c.org/MarkUp/ (HTML, XHTML2)
  http://www.w3.org/QA/2002/04/valid-dtd-list.html (Doctype links)
- DOM specific URLs (as of 2013-05-01)
  http://www.saxproject.org/ (current project home)
  http://www.cafeconleche.org/books/xmljava/chapters/index.html (book)
  http://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html (Java 7 docs)
  W3C: http://www.w3.org/2003/01/dom2-javadoc/org/w3c/dom/traversal/DocumentTraversal.html
  Apache Xerces2: http://xerces.apache.org/xerces2-j/javadocs/api/org/w3c/dom/traversal/DocumentTraversal.html
- Sample files installed with BSF4ooRexx
  - bsf4oorexx/samples/DOM

© 2013 Rony G. Flatscher