

BSF4ooRexx

Parse, Analyze and Process XML Documents with DOM (Document Object Model)
and XSLT (Extensible Stylesheet Language Transformation)

Business Programming 2



BSF4ooRexx



NetRexx

Windows
GUIs
(AWT)

Sockets
SSL/TLS

XML
SAX/DOM
JSON

Scripting
AOO/LO
(UNO)

Rexx
Script
Engine

Portable
GUIs
(JavaFX)

Java Web
Server
(Tomcat)

Java Classes
written in Rexx
style



- Text, marked up in HTML

```
<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>
```

Web Browser Output:

Important Heading

This is the first paragraph.

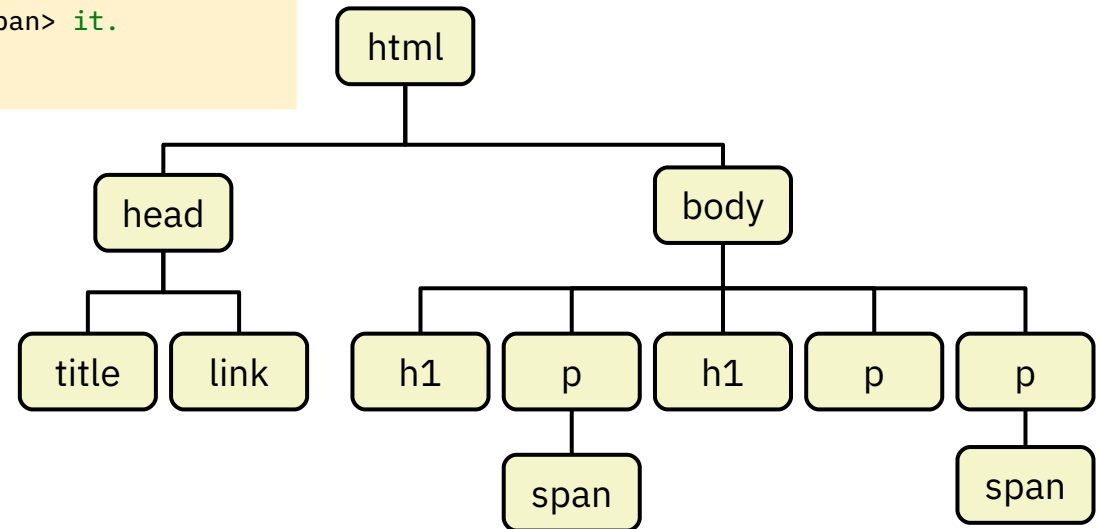
Another Important Heading

Another paragraph.

This is it.

Document Object Model (DOM) – Parse Tree

```
<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>
```



- 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
 - Search for the string "[javadoc 8 org.w3c.dom.Node](#)" or "[javadoc w3c Node](#)"

- 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`

Java DOM Parsers, 3



- 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

Extract Text From Any XHTML Document (1/4)

```

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 ...

```

Extract Text From Any XHTML Document (2/4)

```

... 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
end

::routine processNode    /* processes each node */
  use arg node
  nodeType=node~getNodeTypes  -- 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))

```

Extract Text From Any XHTML Document (3/4)

XHTML

```
<!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>
```

CSS

```
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; }
```

\$ rexx dom_01.rxj example2.html

Output:

```
[
 ]
[
 ]
[This is my HTML file]
[
 ]
[
 ]
[
 ]
[
 ]
[
 ]
[Important Heading]
[
 ]
[
 ]
[This ]
[is]
[ the
    first paragraph.]
[
 ]
[Another Important Heading]
[
 ]
[Another paragraph.]
[
 ]
[This ]
[is]
[ it.]
[
 ]
[
 ]
[
 ]
```

Extract Text From Any XHTML Document (4/4)

- 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`

List Elements in Document Order (1/4)

```

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 ...

```

List Elements in Document Order (2/4)

```

... 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~getNodeTypes -- 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))

```

List Elements in Document Order (3/4)

XHTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml11-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>
```

CSS

```
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; }
```

\$ rexx dom_02.rxj example2.html

Output:

```
[html]
[head]
[title]
[link]
[body]
[h1]
[p]
[span]
[h1]
[p]
[p]
[span]
```

List Elements in Document Order (4/4)

- 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

List Elements Indented in Document Order (1/4)

```

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 ...

```

List Elements Indented in Document Order (2/4)

```

... 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~getNodeTypes      -- 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))

```

List Elements Indented in Document Order (3/4)

XHTML

```
<!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>
```

CSS

```
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; }
```

\$ rexx dom_03.rxj example2.html

Output:

```
[html]
  [head]
    [title]
    [link]
  [body]
    [h1]
    [p]
      [span]
    [h1]
    [p]
    [p]
      [span]
```

List Elements Indented in Document Order (4/4)

- Some 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

List Elements with Text (1/4)

```

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 ...

```

List Elements with Text (2/4)

```

... 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~getNodeTypes      -- 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))

```

List Elements with Text (3/4)

XHTML

```
<!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>
```

CSS

```
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; }
```

\$ rexx dom_04.rxj example2.html

Output:

```
[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.]
--> [
]
--> [
]
```

List Elements with Text (4/4)

- Some 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`, `CDATA_SECTION_NODE`, and `TEXT_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

Little Break ...



- 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](https://www.w3.org/2003/01/dom2-javadoc/org/w3c/dom/traversal/DocumentTraversal)
 - Documentation at:
 - <http://www.w3.org/2003/01/dom2-javadoc/org/w3c/dom/traversal/DocumentTraversal.html>
(2022-12-12)
 - 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

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 "`dom_05.rxj`" with "`dom_02.rxj`" above

List Elements In Document Order (NodeIterator 1/2)

```

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")
::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))

```

List Elements In Document Order (*NodeIterator* 2/2)

XHTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml11-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>
```

CSS

```
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; }
```

\$ rexx dom_05.rxj example2.html

Output:

```
[html]
[head]
[title]
[link]
[body]
[h1]
[p]
[span]
[h1]
[p]
[p]
[span]
```

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 "`dom_06.rxj`" with "`dom_03.rxj`" above

List Elements Indented in Document Order (*TreeWalker* 1/3)

```

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 ...

```

List Elements Indented in Document Order (*TreeWalker* 2/3)

```

... cut ...
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~getNodeName=.show_element then -- make sure only element nodes (not their text 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")

::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))

```

List Elements Indented in Document Order (*TreeWalker* 3/3)

XHTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml11-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>
```

CSS

```
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; }
```

\$ rexx dom_06.rxj example2.html

Output:

```
[html]
  [head]
    [title]
    [link]
  [body]
    [h1]
    [p]
      [span]
    [h1]
    [p]
    [p]
      [span]
```

Little Break ...

- The returned root node object implements the interface classes
 - `org.w3c.dom.Document` extends `org.w3c.dom.Node`
 - `Document` methods, e.g.: `getElementById(...)`, `normalizeDocument()`
 - `org.w3c.dom.Element` extends `org.w3c.dom.Node`
 - `Element` methods, e.g.: `getElementsByTagName(...)`, `getTagName()`
- ➔ Because `Document` and `Element` extend the `Node` interface class all methods and constants from `org.w3c.dom.Node` are available as well!
 - `Node` methods, e.g.: `cloneNode(...)`, `hasChildNodes()`, `getNodeName()`, `getNodeType()`, `getNodeValue()`, `getTextContent()`
- There are many more methods available, once we received the root node

Query Elements and Extract Text (1/4)

```

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

id="xyz1"
node=rootNode~getElementById("xyz1") -- get element node with attribute 'id' set to 'xyz1'
textNode=node~childNodes~item(0)    -- get its text node
say "node with 'id'="pp(id) "nodeName="pp(node~nodeName) "its text:" pp(textNode~nodeValue)
say "---"

tName="p"    -- paragraph tag name
nodes=rootNode~getElementsByTagName("p") -- get all "p" nodes (a NodeList)
say "there are" pp(nodes~getLength) "nodes of type" pp(tName) "in the document:"
say
do i=0 to nodes~getLength-1    -- iterate over the 0-based NodeList
  chNodes=nodes~item(i)~childNodes -- get node's child nodes
  mb=.mutableBuffer~new
  do k=0 to chNodes~getLength-1    -- process child nodes
    call getText chNodes~item(k),mb -- collect all text node's values
  end
  say "  text of '"tName"'-node #" pp(i+1) ":" pp(mb~string)
end
end

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

```

Query Elements and Extract Text (2/4)

```

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

::routine getText      /* collect the node's children text node values */
  use arg node, mb
  -- if pos(node~nodeType, "3 4")>0 then -- type is a text or CDATA section node?
  if wordPos(node~nodeName, "#text #cdata-section")>0 then -- a text or CDATA-section node?
    mb~append(node~nodeValue) -- append the text value

  if node~hasChildNodes then -- collect all the child nodes text if any
  do
    childNodes=node~getChildNodes
    do i=0 to childNodes~getLength-1 -- iterate over the child nodes
      call getText childNodes~item(i), mb
    end
  end
  return

::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))

```

Query Elements and Extract Text (3/4)

XHTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml11-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>
```

CSS

```
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; }
```

```
$ rexx dom_10.rxj example2.html
```

Output:

```
node with 'id'=[xyz1] nodeName=[p] its text: [Another paragraph.]
---
there are [3] nodes of type [p] in the document:

  text of 'p'-node # [1]: [This is the
                        first paragraph.]
  text of 'p'-node # [2]: [Another paragraph.]
  text of 'p'-node # [3]: [This is it.]
```

Query Elements and Extract Text (4/4)

- Some remarks
 - The DOM parse tree and the APIs get document in the Java interface classes
 - [org.w3c.dom.Document](#), e.g. methods [getElementById\(...\)](#), [normalizeDocument\(\)](#)
 - [org.w3c.dom.Element](#) e.g. method [getElementsByTagName\(...\)](#), it extends
 - [org.w3c.dom.Node](#) e.g. methods [hasChildNodes\(\)](#), [getNodeName\(\)](#), [getNodeType\(\)](#)
 - The node types used in the Java DOM parser are retrievable via the Java interface class [org.w3c.dom.Node](#)
 - A *text node* has as node name "[#text](#)" and a node type of [3](#)
 - A *CDATA section node* has a node name "[#cdata-section](#)" and a node type of [4](#)
 - To find the Java documentation on the Internet you can use a search string like "[javadoc 8 w3c Node](#)" or "[javadoc w3c Node](#)"
 - Extracting text can be sometimes simplified with the Node method [getTextContent\(\)](#)

Query Elements and Extract Text: Simpler (1/2)

```

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
-- use DOM to process the parse tree

id="xyz1"
node=rootNode~getElementById("xyz1")-- get element node with attribute 'id' set to 'xyz1'
say "node~getTextContent:" pp(node~getTextContent)
say "---"

tName="p"      -- paragraph tag name
nodes=rootNode~getElementsByTagName("p") -- get all "p" nodes (a NodeList)
say "there are" pp(nodes~getLength) "nodes of type" pp(tName) "in the document:"
say
do i=0 to nodes~getLength-1          -- iterate over the 0-based NodeList
    say "  nodes~item(i)~getTextContent:" pp(nodes~item(i)~getTextContent)
end

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

::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))

```

Query Elements and Extract Text: Simpler (2/2)

XHTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml11-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>
```

```
$ rexx dom_10_simpler.rxj example2.html
```

Output:

```
node with 'id'=[xyz1] nodeName=[p] its text: [Another paragraph.]
---
there are [3] nodes of type [p] in the document:

text of 'p'-node # [1]: [This is the
  first paragraph.]
text of 'p'-node # [2]: [Another paragraph.]
text of 'p'-node # [3]: [This is it.]
```

CSS

```
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; }
```

Little Break ...

Process XML Files with XSLT (Extensible Stylesheet Language Transformation)



- XML based powerful transformation language
 - Allows to transform a XML document into any other XML, HTML or text document
 - Uses W3C's DOM (document object model) standard
 - Selection (of parts) of a XML document using XPATH queries (element names, attribute values)
 - Transformation allows among many other things to apply XSLT functions
- The JRE has built-in support for transforming XML files using XSL stylesheets
 - The REXX program `dom_11.rxj` on the next slide demonstrates how transforming *any* XML file with *any* XSL stylesheet can be carried out

Process XML Files with XSLT

```

/* Process a XML file with the indicated XSL stylesheet. */
parse arg xmlFile xslFile    -- get filenames
signal on syntax
factory = bsf.loadClass("javax.xml.transform.TransformerFactory")~newInstance
clzStreamSource = bsf.importClass("javax.xml.transform.stream.StreamSource")
transformer = factory~newTransformer( clzStreamSource~new(xslFile) )
xmlSource    = clzStreamSource~new(xmlFile)
streamResult= .bsf~new("javax.xml.transform.stream.StreamResult", .java.lang.System~out)
transformer~transform(xmlSource, streamResult)
exit

```

```

syntax:      -- show Java exception chain to become able to understand some of the Java errors
co=condition('o')    -- get REXX condition information
say ppJavaExceptionChain(co, .true)  -- show Java stacktrace
say "----"
raise propagate      -- reraise condition to let ooRexx handle it

```

```

::requires "BSF.CLS"    -- get ooRexx-Java bridge

```

Process *XHTML* Files with XSLT (1/1)

XHTML

example2.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml11-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>
```

XSL

example2.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <xsl:output omit-xml-declaration="yes" />

  <xsl:template match="node()">
    <xsl:apply-templates select="node()"/>
  </xsl:template>

  <xsl:template match="xhtml:h1">
    element name "<xsl:value-of select="name()"/>": <xsl:value-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```

```
$ rexx dom_11.rxj example2.html example2.xsl
```

Output:

```
element name "h1": Important Heading
element name "h1": Another Important Heading
```

Process XML Files with XSLT (1/2)

XML

rexx.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<creators>
  <rexx>
    <title>REXX</title>
    <author>Cowlishaw, Mike F.</author>
    <country>Great Britain</country>
    <company>IBM</company>
    <since>1979</since>
  </rexx>
  <rexx>
    <title>ooRexx (Open Object REXX)</title>
    <author>McGuire, Rick; Nash, Simon</author>
    <country>USA; Great Britain</country>
    <company>RexxLA (non-profit SIG)</company>
    <since>1988 (IBM), 2005 (RexxLA.org)</since>
  </rexx>
  <rexx>
    <title>NetRexx</title>
    <author>Cowlishaw, Mike F.</author>
    <country>Great Britain</country>
    <company>IBM</company>
    <since>1996 (IBM), 2011 (RexxLA.org)</since>
  </rexx>
  <rexx>
    <title>brexx</title>
    <author>Vlachoudis, Vassilis N.</author>
    <country>Greece/France/Switzerland</country>
    <company>n/a (CERN)</company>
    <since>1992</since>
  </rexx>
  <rexx>
    <title>Regina</title>
    <author>Hessling, Mark; Chistensen, Anders</author>
    <country>Australia; Norway</country>
    <company>SINTEF</company>
    <since>1993 ?</since>
  </rexx>
  <rexx>
    <title>ARexx</title>
    <author>Hawes, William S.</author>
    <country>USA</country>
    <company>Amiga (Commodore)</company>
    <since>1987</since>
  </rexx>
</creators>
```

XSL

rexx.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Overview of Some REXX Interpreters</h2>
        <table border="1" style="background-color: LightGoldenrodYellow; border=1; ">
          <tr style="background-color: BurlyWood">
            <th>Title</th>
            <th>Author</th>
            <th>Country</th>
            <th>Since</th>
          </tr>
          <xsl:for-each select="creators/rexx">
            <xsl:sort select="since"/>
            <tr>
              <td><xsl:value-of select="title" /></td>
              <td><xsl:value-of select="author" /></td>
              <td><xsl:value-of select="country" /></td>
              <td><xsl:value-of select="since" /></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Process XML Files with XSLT (2/2)

```
$ rexx dom_11.rxj rexx.xml rexx.xsl
```

Output:

```
<html>
  <body>
    <h2>Overview of Some REXX Interpreters</h2>
    <table border="1" style="background-color: LightGoldenrodYellow; border=1; ">
      <tr style="background-color: BurlyWood">
        <th>Title</th><th>Author</th><th>Country</th><th>Since</th>
      </tr>
      <tr>
        <td>REXX</td><td>Cowlishaw, Mike F.</td><td>Great Britain</td><td>1979</td>
      </tr>
      <tr>
        <td>ARexx</td><td>Hawes, William S.</td><td>USA</td><td>1987</td>
      </tr>
      <tr>
        <td>ooRexx (Open Object REXX)</td><td>McGuire, Rick; Nash, Simon</td><td>USA; Great Britain</td><td>1988 (IBM), 2005 (RexxLA.org)</td>
      </tr>
      <tr>
        <td>brexx</td><td>Vlachoudis, Vassilis N.</td><td>Greece/France/Switzerland</td><td>1992</td>
      </tr>
      <tr>
        <td>Regina</td><td>Hessling, Mark; Chistensen, Anders</td><td>Australia; Norway</td><td>1993 ?</td>
      </tr>
      <tr>
        <td>NetRexx</td><td>Cowlishaw, Mike F.</td><td>Great Britain</td><td>1996 (IBM), 2011 (RexxLA.org)</td>
      </tr>
    </table>
  </body>
</html>
```

Overview of Some REXX Interpreters

Title	Author	Country	Since
REXX	Cowlishaw, Mike F.	Great Britain	1979
ARexx	Hawes, William S.	USA	1987
ooRexx (Open Object REXX)	McGuire, Rick; Nash, Simon	USA; Great Britain	1988 (IBM), 2005 (RexxLA.org)
brexx	Vlachoudis, Vassilis N.	Greece/France/Switzerland	1992
Regina	Hessling, Mark; Chistensen, Anders	Australia; Norway	1993 ?
NetRexx	Cowlishaw, Mike F.	Great Britain	1996 (IBM), 2011 (RexxLA.org)

- 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 !

- DOM specific URLs (2023-01-16)
 - <http://www.saxproject.org/>
 - <http://www.cafeconleche.org/books/xmljava/chapters/index.html>
 - <http://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html>
 - 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>
- Tutorials, e.g. (2023-01-16)
 - XML: <https://www.w3schools.com/xml/default.asp>
 - XPATH: https://www.w3schools.com/xml/xpath_intro.asp
 - XML DOM: https://www.w3schools.com/xml/dom_intro.asp
 - XSLT: https://www.w3schools.com/xml/xsl_intro.asp
- Sample files installed with BSF4ooRexx
 - BSF4ooRexx850/samples/SAX
 - BSF4ooRexx850/samples/DOM