

BSF4ooRexx

Parse, Analyze and Process HTML Documents with **jsoup** (a Java DOM Parser)

Business Programming 2



BSF4ooRexx



NetRexx

Windows
GUIs
(AWT)

Sockets
SSL/TLS

XML
SAX/DOM
JSON

Scripting
A00/LO
(UNO)

Rexx
Script
Engine

Portable
GUIs
(JavaFX)

Java Web
Server
(Tomcat)

Java Classes
written in Rexx
style



Markup Language

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

Web Browser Output:

Important Heading

This is the first paragraph.

Another Important Heading

Another paragraph.

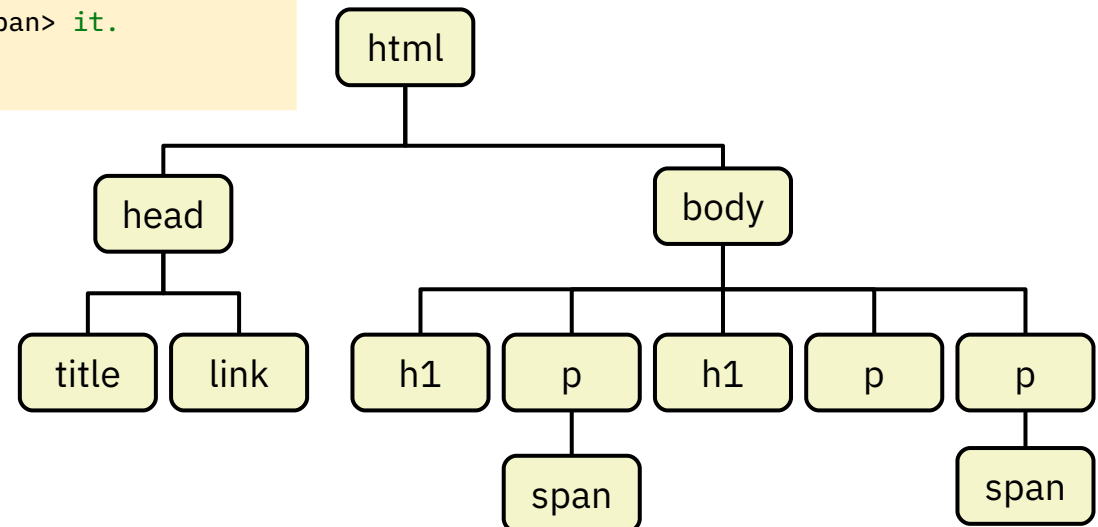
This is it.





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>
```



Java `jsoup`

Brief Overview



- `XML-DOM` parsers depend on valid marked up `XML` text
 - No XML tool can be therefore applied to HTML text or "tag soup" HTML text
- "`jsoup`" – an open-source `Java-DOM` parser for `HTML` and `XHTML`
 - Concepts and acquired skills from XML can be applied, makes it easy to exploit
 - Parses `HTML 5`
 - Parses "tag soup" `HTML`, i.e. `HTML` text with improperly used (at a wrong position) or even undefined tags
 - Can be used to
 - Create proper `HTML` text from "tag soup" (markup with errors) `HTML` text
 - Create proper `xhtml` text from any `HTML` text
 - Change/rewrite `HTML` text
 - ... and much more like "web scraping" or "data wrangling"

How to Get and Setup

- External Java class library/Java archive
 - Homepage: <https://jsoup.org>
 - Download: <https://jsoup.org/download>
 - After download make the external Java archive (e.g. 2024-01-12 jsoup-1.17.2.jar) available to Java either by
 - Adding its full path to the CLASSPATH environment variable
 - Easier, if you have BSF4ooRexx850 installed, proceed as follows:
 - in your home directory (%USERPROFILE% on Windows, \$HOME on Unix) create the subdirectories exactly spelled out as BSF4ooRexx and therein the subdirectory lib and copy the external Java class library into BSF4ooRexx/lib
 - Alternatively copy the external Java archive into BSF4ooRexx850's installation directory and there into its lib subdirectory, however note:
 - Whenever BSF4ooRexx850 gets uninstalled also all external Java archives will get removed such that you would have to copy them again, after a new version of BSF4ooRexx850 gets installed
 - Jsoup's cookbook: <https://jsoup.org/cookbook/>

How To Parse HTML and XML Files, 1

- `org.jsoup.Jsoup` class has various `parse(...)` methods
 - Strings with the marked up text
 - `java.io.File` (local files) and `java.net.URL` (`http` and `https` URLs)
 - Need to create instances from these Java classes to supply as arguments to `parse(...)`
- Ease debugging of Java exceptions while developing
 - The reported Java exception may be ultimately caused by other Java exceptions
 - `BSF4ooRexx850` only shows the reported Java exception
 - To ease debugging
 - Intercept syntax conditions in your programs
 - Use `BSF.CLS'` public routine `ppJavaExceptionChain(condition[,.true])`
 - This will show the chain of Java exceptions
 - The second argument if `.true` will display the Java thread stack of the last Java exception

How To Parse HTML and XML Files, 2

- Jsoup's `parse(...)` will return the root element as an instance of the `org.jsoup.nodes.Document` class which is a subclass of
 - `org.jsoup.nodes.Element` which is a subclass of
 - `org.jsoup.nodes.Node`
 - Use the JavaDocs to learn about all available methods, e.g. by searching for "javadoc Jsoup" or "javadoc jsoup Document" and the like

Extract Text from HTML and XML Documents

- Removes all tags and extracts the text of any [HTML](#) and [XML](#) document
- After parsing the root element can be used to
 - Get the *body* element
 - And fetch its text using one of the [org.jsoup.nodes.Element](#) methods *text()*, *wholeText()* or *wholeOwnText()*

Extract Text From Any [X]HTML Document (1/2)

```

/* purpose: demonstrate how to extract the text from a xml file using DOM */
parse arg source      -- can be html, xml; a local file or URL
url? = source ~startsWith("http")  -- do we need to parse a URL?
timeout = 10000      -- when loading URL timeout: 10 sec (10000 msec)
signal on syntax     -- activate syntax condition handling
if url? then src=.bsf~new("java.net.URL",source)  -- create URL object
             else src=.bsf~new("java.io.File",source)  -- create File object

clzJsoup = bsf.importClass("org.jsoup.Jsoup")  -- import Jsoup class
if url? then rootNode = clzJsoup~parse(src,timeout)
             else rootNode = clzJsoup~parse(src)

body = rootNode~body  -- get body element
say "--- body~text:"  -- demonstrate text() method
say pp(body~text)    -- all text normalized (removing whitespace)
say "--- body~wholeText:"  -- demonstrate wholeText() method
say pp(body~wholeText)  -- all text (with whitespace)
say "rootNode~nodeName:" pp(rootNode~nodeName) "body~nodeName:" pp(body~nodeName)
exit

syntax:  -- label for syntax conditions
  co=condition("object")  -- get directory with condition information
  say ppJavaExceptionChain(co,.true)  -- show Java exception chain
  say "---"
  raise propagate  -- let ooRexx process condition as well

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

```

Extract Text From Any [X]HTML Document (2/2)

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

```
rexx jsoup_01.rxj example2.html
```

Output:

```
--- body~text:
[Important Heading This is the first paragraph. Another Important Heading Another paragraph. This is it.]
--- body~wholeText:
[
  Important Heading
  This is the
    first paragraph.
  Another Important Heading
  Another paragraph.
  This is it.
]
```

- Using method `getAllElements()` of `org.jsoup.nodes.Element`
 - Returns an instance of the `org.jsoup.select.Elements` which subclasses `java.util.ArrayList` which implements the `java.lang.Iterable` interface class
 - BSF4ooRexx supports iterating over Java classes that implement the `java.lang.Iterable` interface with "do ... over" which simplifies coding in ooRexx
- Using method `children()` of `org.jsoup.nodes.Element`
 - Returns an instance of the `org.jsoup.select.Elements` which subclasses `java.util.ArrayList` which implements the `java.lang.Iterable` interface class
 - BSF4ooRexx supports iterating over Java classes that implement the `java.lang.Iterable` interface with "do ... over" which simplifies coding in ooRexx
- The root element's name in jsoup is: `#document`

List Elements in Document Order (1/2)

```
/* purpose: demonstrate how to list the element names in document order */
parse arg source      -- can be html, xml; a local file or URL
url? = source ~startsWith("http")  -- do we need to parse a URL?
timeout = 10000      -- when loading URL timeout: 10 sec (10000 msec)
if url? then src=.bsf~new("java.net.URL",source)  -- create URL object
           else src=.bsf~new("java.io.File",source)  -- create File object

clzJsoup = bsf.importClass("org.jsoup.Jsoup")  -- import Jsoup class
if url? then rootNode = clzJsoup~parse(src,timeout)
           else rootNode = clzJsoup~parse(src)

do el over rootNode~getAllElements  -- get all Element nodes
  say pp(el~nodeName)
end

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

List Elements in Document Order (2/2)

```
<!DOCTYPE HTML>
<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="a9876">This <span class="verb">is</span> it.
  </body>
</html>
```

```
rexx jsoup_02.rxj example2.html
```

Output:

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

List Elements Indented in Document Order (1/2)

```

/* purpose: demonstrate how to list the element names in document order */
parse arg source      -- can be html, xml; a local file or URL
url? = source ~startsWith("http")  -- do we need to parse a URL?
timeout = 10000      -- when loading URL timeout: 10 sec (10000 msec)
if url? then src=.bsf~new("java.net.URL",source)  -- create URL object
           else src=.bsf~new("java.io.File",source)  -- create File object

clzJsoup = bsf.importClass("org.jsoup.Jsoup")  -- import Jsoup class
if url? then rootNode = clzJsoup~parse(src,timeout)
           else rootNode = clzJsoup~parse(src)

call followNode rootNode, 0

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

::routine followNode  -- walks the document tree recursively
  use arg node, level
  call processNode node, level  -- process received node
  do childNode over node~children
    call followNode childNode, level+1  -- recurse
  end

::routine processNode  -- processes each node
  use arg node, level
  say "  " ~copies(level) || pp(node~nodeName)

```

List Elements Indented in Document Order (2/2)

```
<!DOCTYPE HTML>
<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="a9876">This <span class="verb">is</span> it.
  </body>
</html>
```

```
rexx jsoup_03.rxj example2.html
```

Output:

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

List Elements with Text Indented in Document Order (1/2)

```

/* purpose: demonstrate how to list the element names in document order */
parse arg source      -- can be html, xml; a local file or URL
url? = source ~startsWith("http")  -- do we need to parse a URL?
timeout = 10000      -- when loading URL timeout: 10 sec (10000 msec)
if url? then src=.bsf~new("java.net.URL",source)  -- create URL object
           else src=.bsf~new("java.io.File",source)  -- create File object

clzJsoup = bsf.importClass("org.jsoup.Jsoup")  -- import Jsoup class
if url? then rootNode = clzJsoup~parse(src,timeout)
           else rootNode = clzJsoup~parse(src)

call followNode rootNode, 0

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

::routine followNode  -- walks the document tree recursively
  use arg node, level
  call processNode node, level  -- process received node
  do childNode over node~children
    call followNode childNode, level+1  -- recurse
  end

::routine processNode  -- processes each node
  use arg node, level
  say "  "~copies(level) || pp(node~nodeName)
  if node~hasText then  -- if node has text, show it indented as well
    say "    "~copies(level) || "-->" pp(node~wholeOwnText)

```


List Elements with Text Indented in Document Order (2/2)

```
<!DOCTYPE HTML>
<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="a9876">This <span class="verb">is</span> it.
  </body>
</html>
```

```
rexx jsoup_04.rxj example2.html
```

Output:

```
[#document]
--> [
]
  [html]
  --> [
]
    [head]
    --> [
]
      [title]
      --> [This is my HTML file]
      [link]
      [body]
      --> [
]
        [h1]
        --> [Important Heading]
... continued on the right ...
```

```
      [p]
      --> [This the
first paragraph.
]
        [span]
        --> [is]
        [h1]
        --> [Another Important Heading]
        [p]
        --> [Another paragraph.
]
          [p]
          --> [This it.
]
            [span]
            --> [is]
```

Get and Show All Links (1/2)

```
/* purpose: demonstrate how to fetch and list all links */
parse arg source      -- can be html, xml; a local file or URL
url? = source ~startsWith("http")  -- do we need to parse a URL?
timeout = 10000      -- when loading URL timeout: 10 sec (10000 msec)
if url? then src=.bsf~new("java.net.URL",source)  -- create URL object
           else src=.bsf~new("java.io.File",source)  -- create File object

clzJsoup = bsf.importClass("org.jsoup.Jsoup")  -- import Jsoup class
if url? then rootNode = clzJsoup~parse(src,timeout)
           else rootNode = clzJsoup~parse(src)

allLinks = rootNode~select("a")  -- get all a-elements
len=length(allLinks~size)
say "there are" pp(allLinks~size) "links"
do counter c link over allLinks
  -- say "#" c~right(len)":" pp(link~absURL('href')) "for" pp(link~text)
  say "#" c~right(len)":" pp(link~attr('href')) "for" pp(link~text)
end

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

Get and Show All Links (2/2)

```
rexex jsoup_05.rxj http://www.RexxLA.org
```

Output (as of 2023-08-30):

```
there are [32] links
# 1: [/] for []
# 2: [/cookie.srsp] for [Cookie Details]
# 3: [#] for [Accept Cookies]
# 4: [javascript:void(0)] for [-]
# 5: [/index.srsp] for [RexxLA Home]
# 6: [/rexkla/about.srsp] for [About RexxLA]
# 7: [/rexklang/index.srsp] for [About Rexx]
# 8: [/community.srsp] for [Community]
# 9: [/members/index.rsp] for [Members]
# 10: [/events/symposium.rsp] for [Symposium]
# 11: [/rexkla/contact.rsp] for [Contact Us]
# 12: [/news/announce-ooRexx-5.0.html] for [Announcement]
# 13: [/rexkla/about.srsp] for [READ MORE]
# 14: [/members/index.rsp?action=join] for [JOIN FOR FREE]
# 15: [https://donorbox.org/donations-to-rexx-language-association] for []
# 16: [/products.srsp] for [About our products]
# 17: [/rexkla/contact.rsp] for [Contact Us]
# 18: [/members/index.rsp?action=login] for [Member Login]
# 19: [/events/symposium.rsp] for [Rexx Symposia]
# 20: [/rexklang/index.srsp] for [Rexx Language]
# 21: [/rexklang/history/index.srsp] for [Rexx History]
# 22: [/rexkla/privacy.srsp] for [Privacy Policy]
# 23: [https://sourceforge.net/projects/ooorexx/files/] for [Download ooRexx]
# 24: [https://sourceforge.net/projects/regina-rexx/files/] for [Download Regina]
# 25: [http://www.netrexx.org/downloads.nsp] for [Download NetRexx]
# 26: [https://sourceforge.net/projects/bsf4ooorexx/files/] for [Download BSF4ooRexx]
# 27: [https://www.rexkla.org/] for [The Rexx Language Association]
# 28: [https://regina-rexx.sourceforge.io/] for [Regina Rexx]
# 29: [https://www.youtube.com/results?search_query=rexx+language%5C] for []
# 30: [https://www.facebook.com/The-Rexx-Language-Association-444234855698010/] for []
# 31: [https://twitter.com/search?q=rexkla&src=typd] for []
# 32: [https://www.facebook.com/ooRexx/] for []
```

Create Proper XHTML File (1/2)

```
/* purpose: demonstrate how to save a file as xhtml */
parse arg source      -- can be html, xml; a local file or URL
url? = source ~startsWith("http")  -- do we need to parse a URL?
timeout = 10000      -- when loading URL timeout: 10 sec (10000 msec)
if url? then src=.bsf~new("java.net.URL",source)  -- create URL object
           else src=.bsf~new("java.io.File",source)  -- create File object

clzJsoup = bsf.importClass("org.jsoup.Jsoup")  -- import Jsoup class
if url? then rootNode = clzJsoup~parse(src,timeout)
           else rootNode = clzJsoup~parse(src)
  -- set xhtml-mode: restrict entities to xhtml only
outputSettings=rootNode~outputSettings  -- get output settings
  -- make sure xhtml entities only
clzEntities=bsf.loadClass("org.jsoup.nodes.Entities")
outputSettings~escapeMode(clzEntities~EscapeMode~xhtml)
outputSettings~syntax(OutputSettings~Syntax~xml)
if url? then outFilename = "tmpFileForUrl.xhtml"
           else outFilename = source".xhtml"
call charOut outFilename, rootNode~html  -- render as xhtml and save

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

Create Proper XHTML File (2/2)

```
<!DOCTYPE HTML>
<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="a9876">This <span class="verb">is</span> it.
  </body>
</html>
```

```
rexx jsoup_06.rxj example2.html
```

Output:

```
<!DOCTYPE html>
<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>
```

- Parsing any marked up text ([X]HTML) possible
 - [jsoup](#) is a powerful Java class library packed as a Java archive ("jar"-file)
 - Versatile, no need for DTD (Document Type Definition)
 - Because of Java, runs on all important operating systems without a change
- Powerful, versatile, still: quite easy to take advantage of
- Can handle "tag soup" marked up text
- Can create proper xhtml renderings from any marked up text!
- Easy to exploit from ooRexx with the ooRexx-Java bridge BSF4ooRexx !

Further Information



- Jsoup
 - jsoup home page: <<https://jsoup.org>>
- HTML 5 DOM specific URLs (2023-01-26)
 - W3C's HTML 5, DOM: <<https://dom.spec.whatwg.org/>>
- Wikipedia
 - jsoup: <<https://en.wikipedia.org/wiki/Jsoup>>
 - Tag soup: <https://en.wikipedia.org/wiki/Tag_soup>
 - Web scraping: <https://en.wikipedia.org/wiki/Web_scraping>
 - Data wrangling: <https://en.wikipedia.org/wiki/Data_wrangling>
- Tutorials, e.g. (2023-01-26)
 - jsoup cookbook: <<https://jsoup.org/cookbook/>>
 - Tutorialspoint: <<https://www.tutorialspoint.com/jsoup/index.htm>>
 - "Parsing HTML in Java with Jsoup": <<https://www.baeldung.com/java-with-jsoup>>
 - "Parsing and Extracting HTML with Jsoup": <<https://howtodoinjava.com/java/library/complete-jsoup-tutorial/>>