Some notes about implementing ajp for RFC9535 with ixml

ajp – A JSONPATH Processor

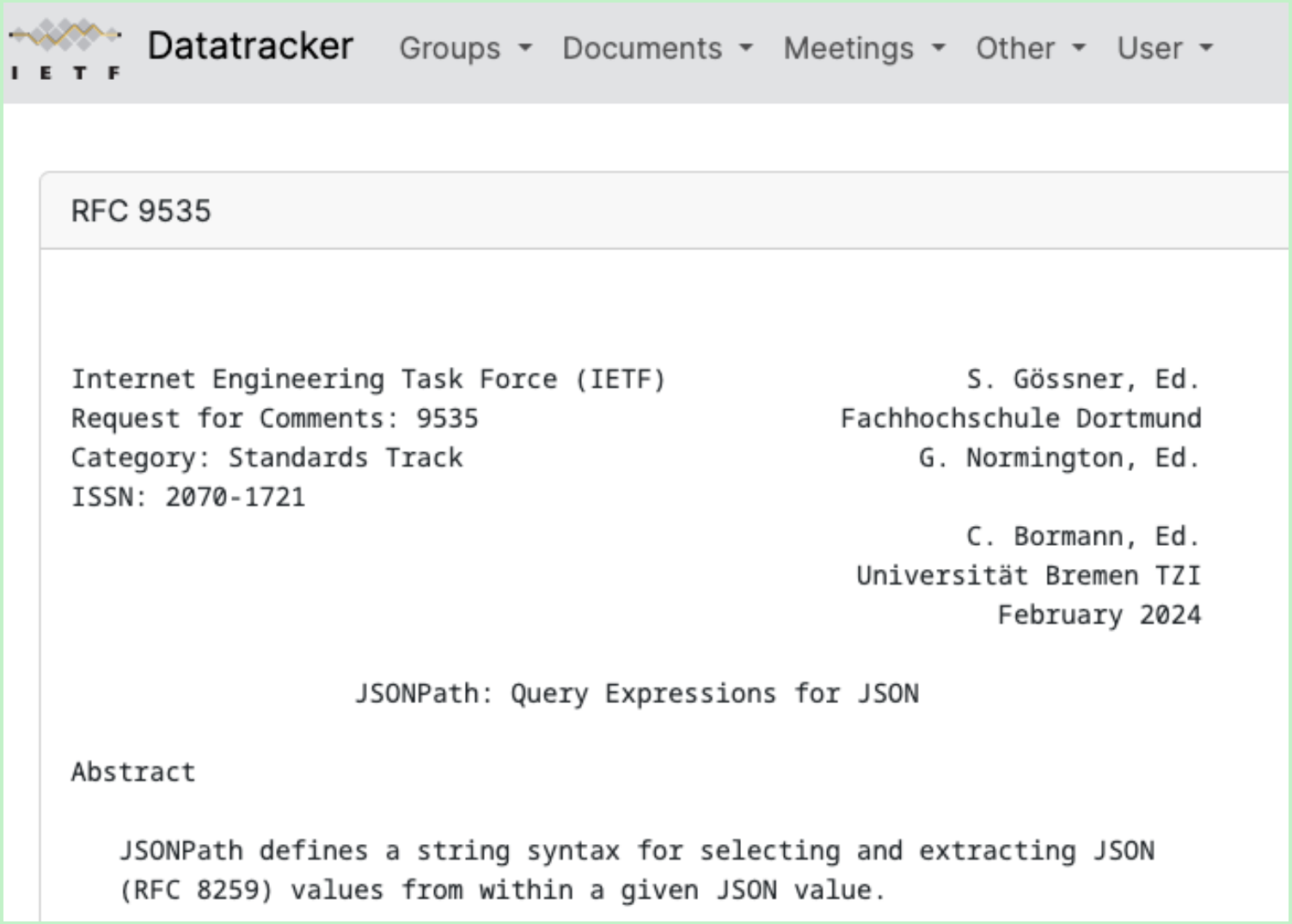
https://github.com/xmljacquard/ajp

Alan Painter

Presented at the 1st International Symposium on iXML

Thursday, 26 February 2026

RFC9535 – JSONPATH – appeared Feb 2024



Datatracker Groups Documents Meetings Other User

RFC 9535

Internet Engineering Task Force (IETF)
Request for Comments: 9535
Category: Standards Track
ISSN: 2070-1721

S. Gössner, Ed.
Fachhochschule Dortmund
G. Normington, Ed.

C. Bormann, Ed.
Universität Bremen TZI
February 2024

JSONPath: Query Expressions for JSON

Abstract

JSONPath defines a string syntax for selecting and extracting JSON (RFC 8259) values from within a given JSON value.

- A standard for JSONPATH after around 50 implementations were developed.

https://datatracker.ietf.org/doc/rfc9535/

RFC9535 – ABNF grammar at its heart (1)

Datatracker

```
Appendix A.  Collected ABNF Grammars

   This appendix collects the ABNF grammar from the ABNF passages used
   throughout the document.

   Figure 2 contains the collected ABNF grammar that defines the syntax
   of a JSONPath query.

   jsonpath-query      = root-identifier segments
   segments            = *(S segment)

   B                   = %x20 /    ; Space
                         %x09 /    ; Horizontal tab
                         %x0A /    ; Line feed or New line
                         %x0D      ; Carriage return
   S                   = *B        ; optional blank space
   root-identifier     = "$"
   selector            = name-selector /
                         wildcard-selector /
```

- Around 150 lines of ABNF to describe the query grammar in IETF ***RFC5234*** (a.k.a ***STD68***).

https://datatracker.ietf.org/doc/std68/

https://datatracker.ietf.org/doc/rfc5234/

RFC9535 – ABNF grammar at its heart (2)

[Jsonpath] Re: Question: Is there a list of conformant RFC9535 implementations?

Glyn Normington <glyn.normington.work@gmail.com> | Tue, 07 October 2025 10:51 UTC | Sho

Hi Alan

I took a quick look at your implementation. I'm pleased it builds on the ABNF of the RFC as this is likely to make it conform to the RFC syntax and it also provides another validation of the ABNF.

I wasn't aware of ixml, but it's an interesting approach albeit with some chunky dependencies.

Congratulations on passing the compliance tests too.

All the best,
Glyn

- Remarks from Glyn Normington, one of the editors of ***RFC9535***, on using *ixml* for the implementation.

https://mailarchive.ietf.org/arch/browse/jsonpath/?q=invisible#

RFC9535 ABNF grammar versus ixml grammar (1)

ABNF

```
jsonpath-query      = root-identifier segments
root-identifier     = "$"
segments            = *( S segment )
segment             = child-segment / descendant-segment
child-segment       = bracketed-selection /
                      ("." (wildcard-selector / member-name-shorthand))
descendant-segment  = ".." (bracketed-selection /
                            wildcard-selector   /
                            member-name-shorthand)
```

ixml

```
jsonpath-query      = root-identifier , segments .
-root-identifier    = - "$" .
segments            = ( S , segment )* .
segment             = child-segment | descendant-segment .
child-segment       = bracketed-selection                                  |
                      ( - "." , ( wildcard-selector | member-name-shorthand ) ) .
descendant-segment  =   - ".." , ( bracketed-selection      |
                                   wildcard-selector        |
                                   member-name-shorthand ) .
```

RFC9535 ABNF grammar versus ixml grammar (2)

ABNF

```
logical-not-op = "!"
paren-expr = [ logical-not-op S ] "(" S logical-expr S ")"

true       = %x74.72.75.65       ; true
false      = %x66.61.6c.73.65    ; false
null       = %x6e.75.6c.6c       ; null
```

ixml

```
logical-not-op = - "!" .
paren-expr = ( logical-not-op , S )? , -"(" , S , logical-expr , S , -")" .

true       = - "true"  .
false      = - "false" .
null       = - "null"  .
```

Handling some special characters (1)

ABNF

```
escapable    = %x62 /          ; b BS backspace U+0008
               %x66 /          ; f FF form feed U+000C
               %x6E /          ; n LF line feed U+000A
               %x72 /          ; r CR carriage return U+000D
               %x74 /          ; t HT horizontal tab U+0009
               "/"  /          ; / slash (solidus) U+002F
               "\"  /          ; \ backslash (reverse solidus) U+005C
```

ixml

```
-escapable  = BS | FF | LF | CR | HT | SLASH | BACKSLASH .

 BS         = -"b"         .    { \b BS backspace                  U+0008 }
 FF         = -"f"         .    { \f FF form feed                  U+000C }
-LF         = -"n" , +#a   .    { \n LF line feed                  U+000A }
-CR         = -"r" , +#d   .    { \r CR carriage return            U+000D }
-HT         = -"t" , +#9   .    { \t HT horizontal tab             U+0009 }
-SLASH      = -"/" , +#2f  .    { \/ slash (solidus)               U+002F }
-BACKSLASH  = -"\" , +#5c  .    { \\ backslash (reverse solidus)   U+005C }
```

Handling some special characters (2)

ABNF

```
                                        ; see RFC 8259
unescaped    = %x20-21              /    ; omit 0x22 "
               %x23-26              /    ; omit 0x27 '
               %x28-5B              /    ; omit 0x5C \
               %x5D-D7FF            /    ; skip surrogate code points
               %xE000-10FFFF
```

ixml

```
                                        { see RFC 8259 }
-unescaped   = [ #20-#21 ]          |    { omit 0x22 "  }
               [ #23-#26 ]          |    { omit 0x27 '  }
               [ #28-#5B ]          |    { omit 0x5C \  }
               [ #5D-#D7FF ]        |    { skip surrogate code points }
               [ #E000-#10FFFD ]    .    { omit #10FFFE - #10FFFF }
```

https://en.wikipedia.org/wiki/Specials_(Unicode_block)

https://datatracker.ietf.org/doc/rfc9839/

Constructing Messages for Parser Errors

```
let $failed := parse-xml($error_description)/failed
return 'Parsing error in query line ' ||        $failed/line       ||
       ' column '                     ||        $failed/column     ||
       ' unexpected character: '      || "'" || $failed/unexpected || "'" || '.'
```

`$[?2.2]`	*Parsing error in query line 1 column 6 unexpected character: ']'.*
`$[?null]`	*Parsing error in query line 1 column 7 unexpected character: ']'.*
`$[?true || false]`	*Parsing error in query line 1 column 9 unexpected character: '|'.*
`$[?true == false && false]`	*Parsing error in query line 1 column 25 unexpected character: ']'.*
`$[?true == false || false]`	*Parsing error in query line 1 column 25 unexpected character: ']'.*
`$[?false && true == false]`	*Parsing error in query line 1 column 10 unexpected character: '&'.*
`$[?false || true == false]`	*Parsing error in query line 1 column 10 unexpected character: '|'.*
`$[?@==True]`	*Parsing error in query line 1 column 7 unexpected character: 'T'.*

A use-case for ixml round-tripping

`$[?length(@[1, 2])<3]` *ajp:FCT0008: argument 1 of function length() must be a singular query.*

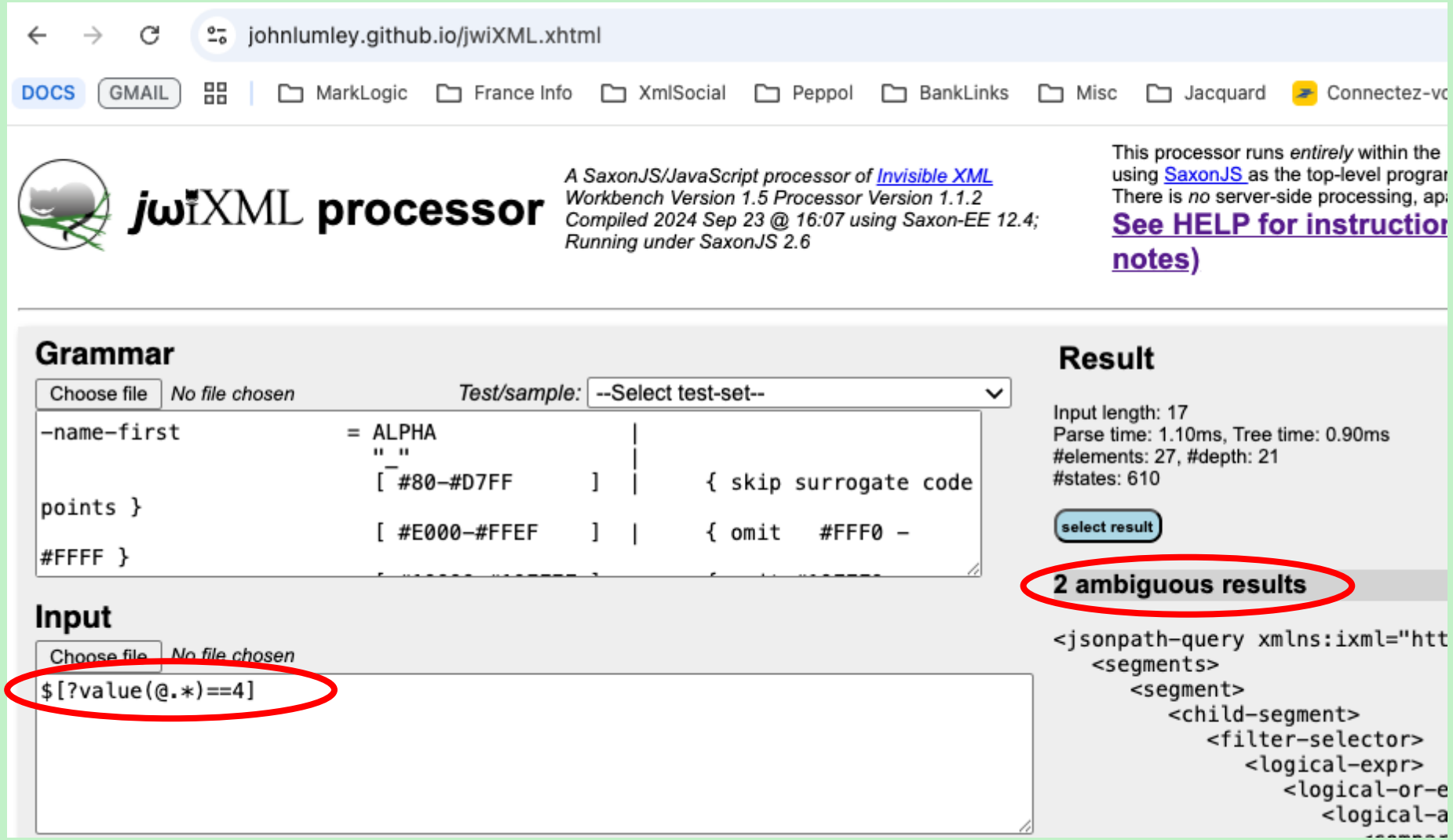
```
<xsl:template match="function-argument[filter-query]" >

  <xsl:param name="functionName" as="xs:string" />

  <xsl:variable name="argumentType" select="ajp:argumentAtPosition($functionName, position())" />

  <xsl:sequence select="if ($argumentType is $VALUE_TYPE and not(ajp:isSingularQuery(filter-query)))
                        then ajp:error('FCT', 8, 'argument '     || position()    ||
                                                 ' of function ' || $functionName || '()’ ||
                                                 ' query: "'     || to-expression(filter-query) '" ' ||
                                                 ' must be a singular query.’)
                        else … " />
</xsl:template>
```

```
function-argument = literal       |
                    filter-query  |  { includes singular-query }
                    logical-expr  |
                    function-expr .
```

A case of ambiguity: $[?value(@.*)==4] (1)

```
function-argument = literal          |
                    filter-query     |  { includes singular-query }
                    logical-expr     |
                    function-expr .

logical-expr      = logical-or-expr                                    .

logical-or-expr   = logical-and-expr , ( S , - "||" , S , logical-and-expr )*   .

logical-and-expr  = basic-expr       , ( S , - "&&" , S , basic-expr      )*   .

-basic-expr       = paren-expr | comparison-expr | test-expr                    .

test-expr         = ( logical-not-op, S )? , ( filter-query | function-expr )   .
```

A case of ambiguity: $[?value(@.*)==4] (2)


johnlumley.github.io/jwiXML.xhtml
DOCS
GMAIL
MarkLogic
France Info
XmlSocial
Peppol
BankLinks
Misc
Jacquard
jwiXML processor
A SaxonJS/JavaScript processor of Invisible XML
Workbench Version 1.5 Processor Version 1.1.2
Compiled 2024 Sep 23 @ 16:07 using Saxon-EE 12.4;
Running under SaxonJS 2.6
This processor runs entirely within the
using SaxonJS as the top-level
There is no server-side processing,
See HELP for instruction
notes)
Grammar
Choose file
No file chosen
Test/sample:
--Select test-set--
-name-first = ALPHA
[#80-#D7FF]
{ skip surrogate code
points }
[#E000-#FFEF]
{ omit #FFF0 -
#FFFF }
Input
Choose file
No file chosen
$[?value(@.*)==4]
Result
Input length: 17
Parse time: 1.10ms, Tree time: 0.90ms
#elements: 27, #depth: 21
#states: 610
select result
2 ambiguous results
<jsonpath-query xmlns:ixml="htt
<segments>

<child-segment>
<filter-selector>
<logical-expr>
<logical-or-e
<logical-a

https://johnlumley.github.io/jwiXML.xhtml

A case of ambiguity: $[?value(@.*)==4] (3)

```
<function-expr>
  <function-name>value</function-name>
  <function-argument>
    <logical-expr>
      <logical-or-expr>
        <logical-and-expr>
          <test-expr>
            <filter-query>
              <rel-query>
                <segments>
                  <segment>
                    <child-segment>
                      <wildcard-selector>*</wildcard-selector>
                    </child-segment>
                  </segment>
                </segments>
              </rel-query>
            </filter-query>
          </test-expr>
        </logical-and-expr>
      </logical-or-expr>
    </logical-expr>
  </function-argument>
</function-expr>
```

```
<function-expr>
  <function-name>value</function-name>
  <function-argument>
    <filter-query>
      <rel-query>
        <segments>
          <segment>
            <child-segment>
              <wildcard-selector>*</wildcard-selector>
            </child-segment>
          </segment>
        </segments>
      </rel-query>
    </filter-query>
  </function-argument>
</function-expr>
```

Can mail me at: **contact@xmljacquard.org**

Thanks!