The Language kvdbJSON

BNF-converter

February 20, 2012

This document was automatically generated by the *BNF-Converter*. It was generated together with the lexer, the parser, and the abstract syntax module, which guarantees that the document matches with the implementation of the language (provided no hand-hacking has taken place).

The lexical structure of kvdbJSON

Literals

Integer literals ⟨*Int*⟩ are nonempty sequences of digits.

String literals ⟨*String*⟩ have the form `"`*x*`"`, where *x* is any sequence of any characters except `"` unless preceded by `\`.

Double-precision float literals ⟨*Double*⟩ have the structure indicated by the regular expression ⟨*digit*⟩ + ‘.’⟨*digit*⟩ + (‘*e*’‘-’?⟨*digit*⟩+)? i.e. two sequences of digits separated by a decimal point, optionally followed by an unsigned or negative exponent.

VarUIdent literals are recognized by the regular expression ‘"’‘$’⟨*upper*⟩(⟨*letter*⟩ | ⟨*digit*⟩ | ‘_’) ∗ ‘"’

UIdent literals are recognized by the regular expression ⟨*upper*⟩(⟨*letter*⟩ | ⟨*digit*⟩ | ‘_’)∗

LIdent literals are recognized by the regular expression ⟨*lower*⟩(⟨*letter*⟩ | ⟨*digit*⟩ | ‘_’)∗

Wild literals are recognized by the regular expression ‘_’⟨*anychar*⟩∗

PrimUUID literals are recognized by the regular expression (‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ |

‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)‘−’(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ |
‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ |
‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ |
‘6’ | ‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ |
‘7’ | ‘8’ | ‘9’)‘−’(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ |
‘7’ | ‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ |
‘8’ | ‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)‘−’(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)‘−’(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ |
‘9’)(‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’)

Reserved words and symbols

The set of reserved words is the set of terminals appearing in the grammar. Those reserved words that consist of non-letter characters are called symbols, and they are treated in a different way from those that are similar to identifiers. The lexer follows rules familiar from languages like Haskell, C, and Java, including longest match and spacing conventions.

The reserved words used in kvdbJSON are the following:

```
false  null  true
```

The symbols used in kvdbJSON are the following:

`{`	`,`	`}`
`"headers"`	`:`	`"body"`
`[`	`]`	`"testReqEmptyHdrs"`
`"testRspEmptyHdrs"`	`"testReqEmptyBody"`	`"testRspEmptyBody"`
`"ask"`	`"answer"`	`"tell"`
`"acknowledge"`	`"response"`	`"request"`
`"getRequest"`	`"fetchRequest"`	`"subscribeRequest"`
`"putRequest"`	`"storeRequest"`	`"publishRequest"`
`"getResponse"`	`"fetchResponse"`	`"subscribeResponse"`
`"putResponse"`	`"storeResponse"`	`"publishResponse"`
`"ok"`	`"notok"`	`/`
`.`		

Comments

There are no single-line comments in the grammar.
There are no multiple-line comments in the grammar.

The syntactic structure of kvdbJSON

Non-terminals are enclosed between ⟨ and ⟩. The symbols ::= (production), | (union) and ϵ (empty rule) belong to the BNF notation. All other symbols are terminals.

⟨*Message*⟩	::=	`{` ⟨*LblReqHeader*⟩ `,` ⟨*LblReqBody*⟩ `}`
	|	`{` ⟨*LblReqBody*⟩ `,` ⟨*LblReqHeader*⟩ `}`
	|	`{` ⟨*LblRspHeader*⟩ `,` ⟨*LblRspBody*⟩ `}`
	|	`{` ⟨*LblRspBody*⟩ `,` ⟨*LblRspHeader*⟩ `}`
⟨*LblReqHeader*⟩	::=	`"headers"` `:` ⟨*ReqHeader*⟩
⟨*LblRspHeader*⟩	::=	`"headers"` `:` ⟨*RspHeader*⟩
⟨*LblReqBody*⟩	::=	`"body"` `:` ⟨*KVDBRequest*⟩
⟨*LblRspBody*⟩	::=	`"body"` `:` ⟨*KVDBResponse*⟩
⟨*ReqHeader*⟩	::=	`[` ⟨*URI*⟩ `,` ⟨*URI*⟩ `,` ⟨*UUID*⟩ `,` ⟨*UUID*⟩ `,` ⟨*ReqJust*⟩ `]`
	|	`[` `{` `"testReqEmptyHdrs"` `:` `null` `}` `]`
⟨*RspHeader*⟩	::=	`[` ⟨*URI*⟩ `,` ⟨*URI*⟩ `,` ⟨*UUID*⟩ `,` ⟨*UUID*⟩ `,` ⟨*RspJust*⟩ `]`
	|	`[` `{` `"testRspEmptyHdrs"` `:` `null` `}` `]`

```
⟨KVDBRequest⟩  ::=  { ⟨AskReq⟩ : ⟨AskReqPacket⟩ }
                 |   { ⟨TellReq⟩ : ⟨TellReqPacket⟩ }
                 |   [ { "testReqEmptyBody" : null } ]

⟨KVDBResponse⟩  ::=  { ⟨AskRsp⟩ : ⟨AskRspPacket⟩ }
                  |   { ⟨TellRsp⟩ : ⟨TellRspPacket⟩ }
                  |   [ { "testRspEmptyBody" : null } ]

⟨AskReqPacket⟩  ::=  { "ask" : ⟨Pattern⟩ }

⟨AskRspPacket⟩  ::=  { "answer" : [ ⟨Pattern⟩ , ⟨Substitution⟩ , ⟨Blob⟩ ] }

⟨TellReqPacket⟩  ::=  { "tell" : [ ⟨Pattern⟩ , ⟨Blob⟩ ] }

⟨TellRspPacket⟩  ::=  { "acknowledge" : ⟨Status⟩ }

⟨ReqJust⟩  ::=  { "response" : null }
            |   { "response" : [ ⟨UUID⟩ ] }

⟨RspJust⟩  ::=  { "request" : null }
            |   { "request" : [ ⟨UUID⟩ ] }

⟨AskReq⟩  ::=  "getRequest"
           |   "fetchRequest"
           |   "subscribeRequest"

⟨TellReq⟩  ::=  "putRequest"
            |   "storeRequest"
            |   "publishRequest"

⟨AskRsp⟩  ::=  "getResponse"
           |   "fetchResponse"
           |   "subscribeResponse"

⟨TellRsp⟩  ::=  "putResponse"
            |   "storeResponse"
            |   "publishResponse"

⟨Status⟩  ::=  "ok"
           |   "notok"
           |   ⟨Integer⟩
           |   ⟨String⟩

⟨Pattern⟩  ::=  ⟨QryTerm⟩

⟨Blob⟩  ::=  ⟨String⟩
```

⟨*Substitution*⟩ ::= `{` ⟨*ListSubstPair*⟩ `}`

⟨*SubstPair*⟩ ::= ⟨*VarUIdent*⟩ `:` ⟨*QryTerm*⟩

⟨*QryTerm*⟩ ::= `{` ⟨*String*⟩ `:` ⟨*QryArray*⟩ `}`

⟨*QryElem*⟩ ::= ⟨*VarUIdent*⟩
| ⟨*QryValue*⟩

⟨*QryValue*⟩ ::= ⟨*QryGrndLit*⟩
| ⟨*QryArray*⟩
| ⟨*QryTerm*⟩

⟨*QryArray*⟩ ::= `[` ⟨*ListQryElem*⟩ `]`

⟨*QryGrndLit*⟩ ::= ⟨*String*⟩
| ⟨*QryNum*⟩
| ⟨*QryBool*⟩
| `null`

⟨*QryBool*⟩ ::= `true`
| `false`

⟨*QryNum*⟩ ::= ⟨*Integer*⟩
| ⟨*Double*⟩

⟨*URI*⟩ ::= ⟨*URIScheme*⟩ `:` ⟨*URIPath*⟩
| `null`

⟨*URIPath*⟩ ::= `/` ⟨*URILocation*⟩ ⟨*URIRelativePath*⟩

⟨*URILocation*⟩ ::= ⟨*URIRoot*⟩ ⟨*URIRsrcLocation*⟩

⟨*URIRsrcLocation*⟩ ::= ⟨*NetLocation*⟩ `:` ⟨*Port*⟩
| ⟨*NetLocation*⟩

⟨*URIRelativePath*⟩ ::= ⟨*URIRoot*⟩ ⟨*ListURIPathElement*⟩

⟨*URIRoot*⟩ ::= `/`

⟨*NetLocation*⟩ ::= ⟨*ListDNSElement*⟩

⟨*URIScheme*⟩ ::= ⟨*LIdent*⟩

```
⟨URIPathElement⟩  ::=  ⟨LIdent⟩

⟨DNSElement⟩  ::=  ⟨LIdent⟩

⟨Port⟩  ::=  ⟨Integer⟩

⟨UUID⟩  ::=  ⟨LIdent⟩
         |   ⟨PrimUUID⟩
         |   null

⟨ListQryElem⟩  ::=  ε
               |    ⟨QryElem⟩
               |    ⟨QryElem⟩ , ⟨ListQryElem⟩

⟨ListSubstPair⟩  ::=  ε
                 |    ⟨SubstPair⟩
                 |    ⟨SubstPair⟩ , ⟨ListSubstPair⟩

⟨ListURIPathElement⟩  ::=  ε
                      |    ⟨URIPathElement⟩
                      |    ⟨URIPathElement⟩ / ⟨ListURIPathElement⟩

⟨ListDNSElement⟩  ::=  ε
                  |    ⟨DNSElement⟩
                  |    ⟨DNSElement⟩ . ⟨ListDNSElement⟩
```