CybOX 3.0 Specification - Pre-Draft

CybOX Patterning – Version 1.0

1. Overview

In order to enhance detection of possibly malicious activity on networks and end-points, a standard language is needed to describe what to look for in an cyber environment. The patterning language allows matching against CybOX Containers w/ a timestamp (e.g., STIX Observed Data Objects) collected by a threat intelligence platform or other similar system so that other analytical tools and systems can be configured to react and handle incidents that might arise.

This first language release is focused on supporting a common set of use cases and therefore allows for the expression of an initial set of patterns that producers and consumers of STIX/CybOX can utilize. As more complex patterns are deemed necessary, the CybOX patterning language will be extended in future releases to improve its effectiveness as an automated detection/remediation method.

1.1. Objects

CybOX does not natively specify timestamps associated with the time that CybOX Objects were observed. Therefore, to allow for the specification of temporal patterns in the STIX context, an object is defined as a STIX Observed Data entity with an embedded CybOX Container and an associated timestamp for when it was observed. The matching of CybOX patterns in the STIX context is expected to be performed against such operands.

1.2. Operators

The CybOX patterning language defines both Boolean and Object operators that provide the ability to express patterns in single term or multi-term expressions as defined in section 2.5.

2. Definitions

The terms defined below are used throughout this document.

Terms	Definitions	Example
whitespace	Any Unicode code point that has WSpace set as a property.	n/a
Object Path	An Object Path defines which property of a CybOX Object (in a CybOX Container) should be matched against as part of a Boolean Expression.	`ipv6-address-object:value`

Boolean Expression	A Boolean Expression is a simple expression consisting of an Object Path and constant separated by a boolean operator.	`user-account-object:value = 'Peter'`
Object Expression	An Object Expression is an expression made of one or more Boolean expressions, or Boolean Expressions joined with object operators. It may be optionally followed by one or more Object Expression Qualifiers.	`ipv4-address-object:value = ‘192.168.1.1’ OR ipv4-address-object:value = '192.168.0.1'`
Object Expression Qualifier	An Object Expression Qualifier is an operator that optionally follows an object expression that provides a restriction on what objects are considered for matching by the object expression.	`file-object:hashes.sha-256 = 'aec070645fe53ee3b3763059376134f058cc337247c978add178b6ccdfb0019f' START '2016-06-01T00:00:00Z' STOP '2016-07-01T00:00:00Z'`

2.1. Pattern Expression

A CybOX *Pattern Expression* is a Unicode string. The pattern **MUST** evaluate to true or false. The object expression **MUST** consist of either a boolean expression, **OR** one or more object expressions joined by object operators. For example, such a pattern could be a boolean expression of an IPv4 address matching against a particular value, or an object expression consisting of two files with different file names OR’d together.

One or more object expression qualifiers **MAY** be provided at the end of an object expression. Boolean expressions joined by object operators are delimited by whitespace. Object expressions may be grouped together with parenthesis (left U+0028 and right U+0029) to change the order of precedence.

As each boolean expression is a singleton, it should be evaluated from left to right in terms of precedence.

Traditional C style comments (starting w/ ‘/*’ U+002f U+002a, and ending with ‘*/’ U+002a U+002f) and C++ style comments (starting w/ ‘//’ U+002f U+002f, ending with either CR U+000d or LF U+000a) **MAY** be present in the pattern string and **MUST** be ignored, and not processed.

When the resulting object expression contains the non-empty set (i.e. there is one or more matching objects), then the pattern has been said to match. How the matching set of objects is used it up to the context of the pattern.

Matching is done by evaluating the pattern. There are Object Expressions which will be matched against a single object (Cybox Container + timestamp). To support matching multiple objects, two object expressions may be combined with an object operator. This indicate that two distinct objects must be matched (or more w/ multiple operators). There may be additional restrictions on on the object expression by time (`WITHIN` or `START`/`STOP`) or that the object expression is repeated multiple times.

Implementation Guidance: The `ALONGWITH` operator does not define what context is required for the two objects to be from, e.g. both objects from the same host. It is expected that we will be able to define this in a future version. In this version it is understood that implementations will likely restrict the contexts from which objects are pulled to match, as needed. It wouldn't make sense to match an indicator containing two file hashes, one on computer A from company B's network with another hash on computer C from company D's network; but currently nothing in this specification restricts or mandates this behavior.

2.2. Constants

The constants specified below are supported in the pattern string. Note that unlike the common datatypes defined in section 2 of the CybOX Core specification, these are independent from a particular serialization, and therefore do not completely match the common datatypes in terms of specification.

Constants	Description
`boolean`	Either the literal characters "true" or "false". Example: `file-object:is_executable = true`
`hex`	The `hex` data type encodes an array of octets (8-bit bytes) as hexadecimal. The string **MUST** consist of an even number of hexadecimal characters, which are the digits '0' through '9' and the letters 'a' through 'f'.
`integer`	A number specified in either decimal (123) or hexadecimal with "0x" prefixed (0x7b). It **MAY** be immediately preceded (no whitespace) by a plus sign (U+002b) or minus sign (U+002d) to indicate a positive or negative number. If a sign is unspecified, the default is positive.
`string`	A string constant is an list of Unicode code points delimited by a single quote (U+0027). The backslash (U+005C) is

	used to escape another backslash, or the single quote delimiter. If a character other than a backslash or single quote character follows the backslash, then the backslash is a normal character and not an escape character and therefore not removed from the string.
`float`	The float data type represents an IEEE 754 double-precision number (e.g., a number with a fractional part). However, because the values ±Infinity and NaN are not representable in JSON, they are not valid values in CybOX.
`timestamp`	TODO: Add text

2.3. Object Expression Qualifiers

Qualifiers	Description
a `REPEATED` *x* `TIMES`	*a* **MUST** be an Object Expression. *a* **MUST** match *x* times, where each match is a different Object.
a `WITHIN` *x* `MILLISECONDS` | `SECONDS` | `MINUTES` | `HOURS` | `DAYS` | `MONTHS` | `YEARS`	*a* **MUST** be an Object Expression. All Objects that match *a* **MUST** occur, or have been observed, within the specified time window. If there are two objects that could be matched by a, that the observation time of the second object is less (not equal to) than the first timestamp + time period.
a `START` *x* `STOP` *y*	*a* **MUST** be an Object Expression. All objects that match a **MUST** have an observation time >= *x* and < *y*. *x* and *y* **MUST** be a timestamp as defined in the STIX 2.0 specification.

2.4. Object Operators

Object operators operate on an object, or set of objects (from another object operator) that were matched by the boolean operators. For example, “a = 'b' **AND** c = 'd' **ALONGWITH** e = 'f'” Will attempt to match an object that has a equal to ‘b’ and c equal to ‘d’ and find a second object that has e equal to ‘f’. As there is no time restriction, this example is bad, and will cause a combinatorial explosion of matches. If you end up receiving 5 objects that match the first clause, and 10 that match the second, you’ll end up w/ 50 matches. Applying a restriction like, “**WITHIN** 5 **SECONDS**” will help prevent the explosion of matches.

Object Operators	Description
a `ALONGWITH` *b*	Both *a* and *b* **MUST** be Object Expressions and **MUST** evaluate to be true on different objects.
a `FOLLOWEDBY` *b*	Both *a* and *b* **MUST** be Object Expressions. *a* **MUST** evaluate to true, and *b* **MUST** also evaluate to true, where *b*’s observation timestamp is greater than, or equal to, *a*’s observation timestamp. All of the objects matched in *a*, their observation timestamp is less than or equal to all of the observation time stamps of the objects matched in *b*. TODO: Clean this up

2.6. Boolean Operators

Boolean Operators	Description
`NOT` *a*	The Boolean Expression *a*’s results are inverted.
a `AND` *b*	Both *a* and *b* **MUST** be Boolean Expressions and **MUST** evaluate to true on the same object.
a `OR` *b*	Both *a* and *b* **MUST** be Boolean Expressions. Either *a* or *b* **MUST** evaluate to true.
a `=` *b*	*a* and *b* **MUST** be equal (transitive), where *a* **MUST** be an Object Path and b **MUST** be a constant of the same type as the property specified in a.
a `!=` *b*	*a* and *b* **MUST** not be equal (transitive), where *a* **MUST** be an Object Path and *b* **MUST** be a constant of the same type as the property specified in *a*.
a `>` *b*	*a* is numerically or lexically greater than *b*, where *a* **MUST** be an Object Path and *b* **MUST** be a constant of the same type as the property specified in *a*.

a `<` *b*	*a* is numerically or lexically less than *b*, where *a* **MUST** be an Object Path and *b* **MUST** be a constant of the same type as the property specified in *a*.
a `<=` *b*	*a* is numerically or lexically less than or equal to *b*, where *a* **MUST** be an Object Path and *b* **MUST** be a constant of the same type as the property specified in *a*.
a `>=` *b*	*a* is numerically or lexically greater than or equal to *b*, where *a* **MUST** be an Object Path and *b* **MUST** be a constant the same type as the property specified in *a*.
a `IN` (*x*,*y*,...)	*a* **MUST** be an Object Path and **MUST** equal a value in the set of x,y,... (transitive). The values in the set **MUST** be constants of the same type, which is one of `binary`, `boolean`, `float`, `integer` or `string`.
a `LIKE` *b*	*a* **MUST** be an Object Path and **MUST** match the pattern specified in *b* where any '%' is 0 or more characters and '_' is any one character. Both strings **MUST** be NFC normalized before comparison.
a `MATCHES` *b*	*a* **MUST** be an Object Path and **MUST** be matched by the pattern specified in *b,* where *b* is a PCRE or PCRE2 compliant regular expression. *b* **MUST** be a slash (U+002f) delimited string. *a* **MUST** be NFC normalized before comparison.
a `CONTAINS` *b*	Both *a* and *b* **MUST** be either ipv4-address-object or ipv6-address-object types or strings that follow the specified value for the CybOX types of IPv4 or IPv6 Address Object. *b* **MUST** be wholly contained within *a*.

2.7. Regular Expressions

Regular expressions **MUST** be defined as PCRE.

3. Operator Precedence

Operators	Associativity
`()`	left to right
`NOT`	right to left
`AND`	left to right
`OR`	left to right
`ALONGWITH` `FOLLOWEDBY`	left to right

4. Comparison

For simple string operators, i.e., "`=`", "`!=`", "`<`", "`>`", "`<=`" and "`>=`", as collation language and method are not able to be specified, a simple code point (binary) comparison **MUST** be used. Unicode normalization **MUST NOT** be performed on the string. This means that combining marks are sorted by their code point, not the NFC normalized value. E.g. 'o' U+006f < 'oz' U+006f U+007a < 'ò' U+006f U+0300 < 'z' U+007a < 'ò' U+00f2. Though Unicode recommends normalizing strings for comparisons, the use of combining marks may be significant, and normalizing by default that would remove this information. NFC normalization is required for other matching operators, e.g., `LIKE` and `MATCHES`.

5. Basic Mathematical Functions That Can Be Applied To Numerics:

`^` : Exponentiation
`+` : Addition
`-` : Subtraction
`/` : Division
`*` : Multiplication
`%` : Modulo

6. CybOX Object Path Syntax

Defined below is the syntax for the abstract specification of CybOX Objects in a CybOX pattern. The following notation is used throughout the definitions below:

Notation	Definition
`<object-type>`	Specifies the type of CybOX Object to match against. This **MUST** be the value of the **type** field defined for the Object.
`<property_name>`	Specifies the name of a CybOX Object property to match against. This **MUST** be a valid property name as specified in the definition of the Object type referenced by the `<object-type>` notation. Properties that are nested, i.e. are children of other properties in a CybOX Object, **MAY** be specified using the syntax `<property_name>.<property_name>`, where the `<property_name>` preceding

	the '.' is the name of the parent property and the one after is the name of the child property.

6.1. Object Properties

Any non-dictionary and non-list property that is directly specified on a CybOX Object.

6.2. Syntax

`<object-type>:<property_name>`

Example

```
file-object:size
```

6.2.1. Nested Object Properties

Any non-dictionary and non-list property that is nested inside of another property on a CybOX Object.

6.2.2. Syntax

`<object-type>:<property_name>.<property_name>`

Example

```
file-object:file_system_properties.file_name
```

6.2.3. List Object Fields

Any property on a CybOX Object that uses the `list` data type.

6.2.4. Syntax

`<object-type>:<property_name[list_index]>.<property_name>`

Where `property_name` **MUST** be the name of an Object property of type `list` and `[list_index]` **MUST** be one of the following:

- An integer in the range of 0..N-1, where N is the length of the list.
- The literal '*', which signifies that matching can be performed against any item in the list. If list_index it out of range, the result of any operation is false.

6.2.5. Dictionary Object Fields

Any property on a CybOX Object that uses the `dictionary` data type.

6.2.6. Syntax

`<object-type>:<property_name>.<key_name>`

Where `<property_name>` **MUST** be the name of an Object property of type `dictionary` and `<key_name>` **MUST** be the name of key in the dictionary.

Examples

```
file-object:hashes.md5
```

```
file-object:extended_properties.raster-image.image_height
```

6.2.7. Object Relationships

Any property on a CybOX Object that uses the `object-ref` data type, either as a singleton or as a list.

Singleton relationships

`<object-type>:<property_name> = {<object-type>:<property_name>}`

Where `<property_name>` **MUST** be the name of an Object property of type `object-ref` and the curly braces ("{" and "}") **MUST** encompass a boolean expression representing the CybOX Object referenced by `<property_name>`.

Lists of relationships

`<object-type>:<property_name[list_index]> = {<object-type>:<property_name>}`

Where `<property_name>` **MUST** be the name of an Object property of `list` of type `object-ref`, `[list_index]` **MUST** meet the constraints for list indices defined in 6.2.3, and the curly braces ("{" and "}") **MUST** encompass a boolean expression representing the CybOX Object referenced by `<property_name[list_index]>`.

6.2.9. One Field Across Multiple Object Types

`*:field_name`

Example

```
*:body
```

7. JSON Serialization

Patterns **MUST** be serialized in JSON as double-quoted strings. As an example:

```
{
  "pattern": "'192.168.0.1/24' CONTAINS ipv4addr-object:value"
}
```

Whitespace (Unicode code points where WSpace=Y) in the pattern string are used to delimit parts of the pattern, including keywords, constants, and field objects. Whitespace characters, including line feeds and carriage returns, between operators **MUST** be

allowed. Multiple whitespace characters in a row **MUST** be treated as a single whitespace character.

8. Implementation

Consumers of CybOX patterning are not required to support and implement the full set of operators specified as part of the language. Rather, they are encouraged to support only the set of operators necessary for their particular use case(s). For example, the vendor of a network intrusion detection system (NIDS) that looks for malicious network traffic would only need to implement the `=` and `IN` operators to match against network connections and IP addresses.

9. Examples:

9.1. Matching a file object with a SHA-256 hash

```
file-object:hashes.sha-256 =
'aec070645fe53ee3b3763059376134f058cc337247c978add178b6ccdfb0019f'
```

9.2. Matching a CIDR containing an ipv4addr-object

```
'192.168.0.1/24' CONTAINS ipv4-address-object:value
```

9.3. Matching an emailaddr-object using a regular expression

```
emailaddr-object:value MATCHES /.+\@ibm\.com$/ ALONGWITH file-object:name MATCHES
/^Final Report.+\.exe$/
```

9.4. Matching a file object with a SHA-256 hash and is a PDF object, simultaneously against the same file

```
file-object:hashes.sha-256 =
'aec070645fe53ee3b3763059376134f058cc337247c978add178b6ccdfb0019f' AND
file-object:mime_type = 'application/x-pdf'
```

9.5. Matching a file object with a SHA-256 hash, and a different file object that has a different SHA-256 hash, against two different files (implied simultaneously on 1 host)

```
file-object:hashes.sha-256 =
'bf07a7fbb825fc0aae7bf4a1177b2b31fcf8a3feeaf7092761e18c859ee52a9c'
 ALONGWITH file-object:hashes.sha-256 =
'aec070645fe53ee3b3763059376134f058cc337247c978add178b6ccdfb0019f'
```

9.6. Matching a file object with a MD5 hash, followed by (temporally) a registry key that matches a value, within 5 minutes

```
file-object:hashes.md5 = '79054025255fb1a26e4bc422aef54eb4' FOLLOWEDBY
win-registry-key-object:key = 'hkey_local_machine\foo\bar' WITHIN 5 MINUTES
```

9.7. Matching three different, but specific users that are observed within 5 minutes, in any order.

```
( user-account-object:value = 'Peter' ALONGWITH user-account-object:value = 'Paul'
ALONGWITH user-account-object:value = 'Mary' ) WITHIN 5 MINUTES
```

9.8. Matching a base64-encoded string in a pcap

```
artifact-object:mime-type = 'application/vnd.tcpdump.pcap' AND artifact-object:payload
MATCHES /Zm9vYmFy/
```

9.9. Matching a base64-encoded string in a network flow payload

```
network-connection-object.extended_properties.flow-extension.source_payload MATCHES
/dGVzdHRlc3R0ZXN0/
```

9.10. Matching a file object with a Windows file path

```
file-object:file_path = 'C:\\Windows\\System32' AND
file-object:file_name = 'foo.dll'
```

9.11. Matching on Windows PE file sections with high entropy

```
file-object:extended_properties.windows-pebinary.sections[*].entropy > 7.0
```

9.12. Matching on a mismatch between File magic number and mime type

```
file-object:mime_type = 'image\\bmp' AND file-object:magic_number = 'ffd8'
```

9.13. Matching on a network connection with a particular destination

```
network-connection-object:dst_refs[*] = {ipv4-address-object.value = 192.168.1.1}
```

10. Open Questions/TODOs

1. Should patterns have a way of specifying the "type" of pattern represented? -- No, this is info on the indicator, or object containing the pattern
    a. E.g., "network pattern", "endpoint pattern", etc.
2. Need to figure out how to represent Actions
3. Need to figure out how to do "ANY OF X matches this regex"

10.1. Suggested examples/use-cases:

1) Match a set of observables that identifies a set of traffic parameters that have occurred in the last 24 hours and return the list of element (IPs, CIDRs...etc) that have been associated with them (i.e. an observation associated with the element)
2) Match all domains that have been observed in traffic sent or being sent from threat actor 'X' and filter which ones of them are based in country 'Y'
3) Match all IPs & Domains observed with http user-agent 'X' and those IPs or Domain are known to be part of a named TTP TTP 'Y'
4) Match all file hashes observed that represent malware name XX that have IOC 'reg-key-modification X' and IOC 'connect to IP Y'

11. Future work

1. Wall clock relative times. e.g., match during/not during business hours only.
1. Extensions (e.g., geoIP use case)
2. Locale related extensions
3. Action-based patterns (if Actions are included in CybOX)

12. ANTLR Grammar

```
grammar CyboxPattern;

startExpression
    :   objectExpression EOF
    ;

objectExpression
    :   left=objectExpression bop=( AND | OR | ALONGWITH | FOLLOWEDBY | EQ)
right=objectExpression (window=timeWindow)?
    |   (NOT)? LPAREN be=objectExpression RPAREN (window=timeWindow)?
    |   (NOT)? booleanExpression (window=timeWindow)?
    ;

timeWindow
    :   START startTime=Timestamp (STOP stopTime=Timestamp)?
    |   WITHIN timespec=IntegerLiteral
(MILLISECONDS|SECONDS|MINUTES|HOURS|DAYS|MONTHS|YEARS)
    ;

booleanExpression
    :   expression booleanOperator
    |   cidrExpression
    ;

cidrExpression
    : cidrLiteral CONTAINS cidrLiteral
    ;

booleanOperator
    :   comparisonOperator | likeOperator | inOperator | regexOperator;

comparisonOperator : cop=(EQ | NEQ | LT | LE | GT | GE) expression ;

likeOperator       : op=LIKE pattern=expression ;

regexOperator    : REGEX literal;

inOperator         : IN LPAREN (expressionList) RPAREN;

expressionList
    :   thisexpr+=expression (COMMA thisexpr+=expression)*
    ;

expression
    :   literal
    |   ObjectPath
    |   LPAREN expression arithmeticOperator RPAREN
    |   expression arithmeticOperator
    |   expression REPEATED repeats=IntegerLiteral TIMES
    ;

arithmeticOperator
    :   op = POWER_OP expression
    |   op = ASTERISK expression
    |   op = DIVIDE expression
    |   op = PLUS expression
```

```
    |   op = MINUS expression
    |   op = MODULO expression
    ;

arguments
    :   args+=expression (COMMA args+=expression)*
    ;

cidrLiteral
    :   StringLiteral
    |   ObjectPath
    ;

literal
    :   (PLUS|MINUS)?IntegerLiteral          #LiteralInteger
    |   (PLUS|MINUS)?IntegerHexLiteral       #LiteralHexInteger
    |   (PLUS|MINUS)?FloatingPointLiteral    #LiteralFloat
    |   StringLiteral                        #LiteralString
    |   (TRUE|FALSE)                         #LiteralBoolean
    |   NULL                                 #LiteralNull
    ;

signedInteger
    :   (PLUS|MINUS)?IntegerLiteral
    ;

////////////////////////////////////////////
// Keywords

AND:  A N D;
ALONGWITH:  A L O N G W I T H ;
OR:  O R;
NOT:  N O T;
FOLLOWEDBY:  F O L L O W E D SPACE B Y ;
LIKE:  L I K E ;
REGEX:  M A T C H E S ;
CONTAINS:  C O N T A I N S ;
LAST:  L A S T ;
IN:  I N;
START:  S T A R T ;
STOP:  S T O P ;
MILLISECONDS:  M I L L I S E C O N D S;
SECONDS:  S E C O N D S;
MINUTES:  M I N U T E S;
HOURS:  H O U R S;
DAYS:  D A Y S;
MONTHS:  M O N T H S;
YEARS:  Y E A R S;
TRUE:  T R U E;
FALSE:  F A L S E;
NULL:  N U L L;

WITHIN:  W I T H I N;
REPEATED:  R E P E A T E D;
TIMES:  T I M E S;

// Timestamp syntax based on Stix 2.0 Timestamp
Timestamp
    :   QUOTE Date Time QUOTE
    ;
```

```
Date
    :   Year HYPHEN Month HYPHEN Day 'T'
    ;

Time
    :   Hours COLON Minutes COLON Seconds 'Z'
    ;

fragment
Year
    :   '2' [0-1] Digit Digit
    ;

fragment
Month
    :   '0' NonZeroDigit
    |   '1' [0-2]
    ;

fragment
Day
    :   '0' NonZeroDigit
    |   [1-2] Digit
    |   '3' [0-1]
    ;

fragment
Hours
    :   [0-1] Digit
    |   '2' [0-3]
    ;

fragment
Seconds
    :   Minutes DOT Digits
    ;

fragment
Minutes
    :   [0-5] Digit
    ;

IntegerLiteral
    :   DecimalNumeral
    ;

IntegerHexLiteral
    :   HexNumeral
    ;

FloatingPointLiteral
    :   DecimalFloatingPointLiteral
    ;

fragment
DecimalFloatingPointLiteral
    :   Digits DOT Digits? ExponentPart?
    |   DOT Digits ExponentPart?
    |   Digits ExponentPart
    |   Digits
    ;
```

```
fragment
DecimalNumeral
    :   '0'
    |   NonZeroDigit (Digits)?
    ;

fragment
Digits
    :   Digit+
    ;

fragment
Digit
    :   '0'
    |   NonZeroDigit
    ;

fragment
NonZeroDigit
    :   [1-9]
    ;

fragment
ExponentPart
    :   ExponentIndicator SignedInteger
    ;

fragment
ExponentIndicator
    :   [eE]
    ;

fragment
HexNumeral
    :   '0' [xX] HexDigits
    ;

fragment
HexDigits
    :   HexDigit*
    ;

fragment
HexDigit
    :   [0-9a-fA-F]
    ;

fragment
SignedInteger
    :   Sign? Digits
    ;

fragment
Sign
    :   [+-]
    ;

StringLiteral
    :   //QUOTE StringCharacters? QUOTE
        QUOTE ( ~'\'' | '\'\'' )* QUOTE
```

```
    ;

fragment
StringCharacters
    :   StringCharacter+
    ;

fragment
StringCharacter
    :   ~['\\]
//    |   EscapeSequence
    ;

fragment
Letter
    :   [a-zA-Z$_]
    ;

fragment
LetterOrDigit
    :   [a-zA-Z0-9$_]
    ;

ObjectPath
    :   SimpleObject
    |   NestedObject
    |   ListObject
    |   ExtObject
    ;

ExtObject
    :   SimpleObject LBRACK StringLiteral RBRACK DOT Identifier
    ;

ListObject
    :   SimpleObject LBRACK (Digits | ASTERISK) RBRACK DOT Identifier
    ;

NestedObject
    :   SimpleObject DOT Identifier
    ;

SimpleObject
    :   Identifier COLON Identifier
    ;

Identifier
    :   '"' (~'"' | '""')* '"'
    |   [a-zA-Z_0-9-]+
    ;

EQ        :   '=' | '==';
NEQ       :   '!=' | '<>';
LT        :   '<';
LE        :   '<=';
GT        :   '>';
GE        :   '>=';

QUOTE     : '\'';
SEMI      : ';' ;
COLON     : ':' ;
```

```
DOT       : '.' ;
COMMA     : ',' ;
RPAREN    : ')' ;
LPAREN    : '(' ;
RBRACK    : ']' ;
LBRACK    : '[' ;
PLUS      : '+' ;
HYPHEN    : MINUS ;
MINUS     : '-' ;
NEGATION  : '~' ;
VERTBAR   : '|' ;
BITAND    : '&' ;
MODULO    : '%' ;
POWER_OP  : '^' ;
DIVIDE    : '/' ;
ASTERISK  : '*';

fragment A:  [aA];
fragment B:  [bB];
fragment C:  [cC];
fragment D:  [dD];
fragment E:  [eE];
fragment F:  [fF];
fragment G:  [gG];
fragment H:  [hH];
fragment I:  [iI];
fragment J:  [jJ];
fragment K:  [kK];
fragment L:  [lL];
fragment M:  [mM];
fragment N:  [nN];
fragment O:  [oO];
fragment P:  [pP];
fragment Q:  [qQ];
fragment R:  [rR];
fragment S:  [sS];
fragment T:  [tT];
fragment U:  [uU];
fragment V:  [vV];
fragment W:  [wW];
fragment X:  [xX];
fragment Y:  [yY];
fragment Z:  [zZ];

fragment SPACE:  ' ';

// Whitespace and comments
//
WS  :  [
\t\v\r\n\u000C\u0085\u00a0\u1680\u2000\u2001\u2002\u2003\u2004\u2005\u2006\u2007\u2008\
u2009\u200a\u2028\u2029\u202f\u205f\u3000]+ -> skip
    ;

COMMENT
    :   '/*' .*? '*/' -> skip
    ;

LINE_COMMENT
    :   '//' ~[\r\n]* -> skip
    ;
```