1 EventFilter Query Language

The following gives a short introduction into a newly proposed query langue to create OPC UA EventFilter. The OPC Foundation specifies OPC UA EventFilter in the OPC UA Specification Part 4:Services. EventFilter consists of a selectClauses that defines a list of values to be returned from a filtered event and a whereClause that defines a ContentFilter that contains criteria which have to be met to receive a notification about the event. The values listed in the selectCaluses are defined as SimpleAttributeOperands, so that the selectClauses consists of one or multiple SimpleAttributeOperands. The ContentFilter is a collection of operators where each of them features a corresponding set of operands.

1.1 Structure of the Query Language

In accordance to the EventFilter definition, the query language is split into a SELECT and a WHERE statement. Since the whereClause can describe a complex structure of operators, the corresponding statement can be rather complex and confusing. To increase the usability of the language, it introduces the FOR statement as an additional element that only occurs within the language. Consequently, the concept of OPC UA EventFilter is not altered. The novelty of the FOR statement are references for operators and operands. As a result, the WHERE statement can be simplified by referencing further operands or operators as elements, which are then resolved within the FOR statement.

References can be used to link elements, either declared in the FOR statement or in the WHERE statement, to one concrete element listed in the FOR statement. While multiple elements can reference one particular element, each reference can only be resolved once. Since ElementOperands point to one ContentFilterElement inside the ContentFilterElementArray, they can be seen as a reference from one operator to another. However, the logic of EventFilters allows such references only for the OR and the AND operator. References that are used to describe ElementOperands can be freely chosen, since they are replaced by the ContentFilterElementArrayIndex when constructing the EventFilter.

The example below should illustrate the logic of such references. Lines 1-4 specify the selectClauses of the filter. Each additional operand of the selectClauses is separated by a comma. The ContentFilter is defined from line 5 on. Its basic structure features an OR operator (line 6), which references a GREATHERTHAN operator with $"ref_1". The corresponding operator is then resolved in line 8 and includes a reference to an operand

($”ref_2”). The concrete operand is then declared in line 9 by dereferencing $”ref_2”:=. As last element, the OR operator has an OFTYPE operator which references a NodeId defined with reference $42.

```
1 :  SELECT
2 :  PATH “/Message”,
3 :  PATH “/0:Severity”,
4 :  PATH “/EventType”
5 :  WHERE
6 :  OR($"ref_1", OFTYPE  $42)
7 :  FOR
8 :  $"ref_1" := PATH "/Severity" GREATERTHAN $"ref_2"
9 :  $"ref_2" := UINT32 10
10 :  $42 := ns = 1; i = 5003
```

Since both, references and ElementOperands specify a reference to either an operand or an operator, they have an equal notation. Both are defined by setting a $ in front of either a number or, a string surrounded by quotation marks. However, they have to be resolved by extending the initial reference with :=, followed by either an operand or operator (see e.g. $”ref_1” declared in line 6 is then resolved in line 8). It is important to consider that references to operands are only allowed for Operators and references to Operators are only allowed by ElementOperands to create valid EventFilters. Additionally, references can be nested, so that an initially referenced operators can contain further references to operands (The OR operator in line 6 references the GREATERTHAN operator in line 8, which in turn references the LiteralOperand in line 9).

1.2 BNF of the Query Language for EventFilter

The BNF used for these examples uses ‘<´ and ‘>´ to mark symbols, ‘[´ and ‘]´ to identify optional paths and ‘|´ to identify alternatives. If the ‘(‘ and ‘)’ symbols are used, it indicates sets. Lastly, the ’{’ and ’}’ symbols indicate that an element occurs zero or multiple times. Expressions surroundet by ’´ have to be matched exactly.

In addition not all elements are resolved down to the smallest element. Such elements are String-, JsonString-, Number, ByteString, Guid and relativePath expressions. The Query Language allows any element to be part of a string except single quotation marks. Quotation marks have to be escaped with ///”. JsonStrings correspond to the OPC UA Json encoding. Since the query language implementation uses the open62541 Json parser, limitations and escape characters from this parser have to be considered within the query language. Currently, the usage of JsonStrings for LiteralOperands is limited to OPC UA Variants. Other elements with restrictions related to open62541 parsers are relative-path-elements, Guids, ByteStrings and NodeIds so that the corresponding escape characters and limitations have to be considered. Lastly, Numbers can be any number, including negative values and numbers with decimal places.

```
<EventFilter> ::= <SelectClauses> <ContentFilter>

<SelectClauses> ::= ’SELECT’ {<SimpleAttributeOperand> ’,’} <SimpleAttributeOperand>

<ContentFilter> ::= <WhereClause> [<ForClause>]

<WhereClause> ::= ’WHERE’ (<Operator> | <ElementReference>)

<ElementReference> ::= ’$’ (′”String”′ | ’Number’)

<Operator> ::=
         <SingleOperandOperator>
        | <TwoOperandsOperator>
        | <BetweenOperator>
        | <InListOperator>
        | <BranchOperator>

<SingleOperandOperator> ::= (’OFTYPE’ | (’ISNULL’ | ’0=’) | (’NOT’ | ’!’)) <Operand>

<TwoOperandsOperator> ::=
        <Operand>
        (
          (’GREATERTHAN’ | ’GT’ | ’>’)
        | (’EQUALS’ | ’EQ’ | ′”==”′)
        | (’LESSTHAN’ | ’LT’ | ’<’)
        | (’GREATEROREQUAL’ | ’GE’ | ′”>=”′)
        | (’LESSOREQUAL’ | ’LE’ | ′”<=”′)
        | (’LIKE’ | ′”<=>”′)
        | (’CAST’ | ’->’)
        | (’BITAND’ | ’&’)
        | (’BITOR’ | ’v’)
        )
        <Operand>

<BetweenOperator> ::= <Operand> ’BETWEEN’ ’[’<Operand>, <Operand>’]’

<InListOperator> ::= <Operand> ’INLIST’ ’[’ {<Operand> ’,’} <Operand>’]’

<BranchOperator> ::= (’AND’ | ’OR’) ’(’<Operand> ’,’ <Operand>’)’
```

```
<Operand> ::=
             <ElementOperand>
           | <SimpleAttributeOperand>
           | <LiteralOperand>
           | <ElementReference>

<ElementOperand> ::= <ElementReference>

<SimpleAttributeOperand> ::=
           [’TYPEID’ <NodeId>]
           ’PATH’ ′”relative path”′
           [’ATTRIBUTE’ ’Number’]
           [’INDEX’ ’IndexRange’]

<LiterlOperand> ::= ’JsonString’ | <Literal>

<Literal> ::=
           ( ([’STRING’] ′”String”′ )
          | (’BOOL’ (’True’ | ’False’ | ’false’ | ’true’ | ’1’ | ’0’) | (’True’ | ’False’ | ’false’ | ’true’))
          | [’INT64’] "Number"
          | (’INT16’ | ’UINT16’ | ’UINT32’ | ’INT32’ | UINT64 ) ’Number’
          | [’DOUBLE’] ’floating point’
          | FLOAT ’floating point’
          | [’NODEID’] <NodeId>
          | ’SBYTE’ ’-’, ’Number’
          | ’BYTE’ ’Number’
          | ’TIME’ ’TimeValueString’
          | ’GUID’ ’GUID-String’
          | ’BSTRING’ ′”String”′
          | ’STATUSCODE’ (′”String”′| ’Number’)
          | ’EXPNODEID’ ’"ExpandedNodeIdString"’
          | [’QNAME’] ’"relative-path-element"’
          | ’LOCALIZED’ ’"String"’)

<ForClause> ::= ’FOR’ <ReferencedElement> {<ReferencedElement>}

<ReferencedElement> ::= ElementReference ’:=’ (<Operand> | <Operator>)

<NodeId> ::=
```

['ns=' 'Number' ';']

('s=' '"String"'

| 'i=' 'Number'

| 'b=' '"ByteString"'

| 'g=' 'Guid')

1.3 Simple EventFilter Query Examples

The following examples illustrate how to build the EventFilter from the open62541 EventFilter examples (`https://github.com/open62541/open62541/blob/master/examples/events/client_eventfilter.c`) with the proposed query language.

1.3.1 SelectClauses

Since all considered EvenFilter have identical SelectClauses, the corresponding statetement is only explained once. Each selectClauses consists of a list of one or multiple SimpleAttributeOperands which are separated by a comma:

$$\begin{aligned} &\text{SELECT}\\ &\text{PATH "/Message",}\\ &\text{PATH "/0:Severity",}\\ &\text{PATH "/EventType"} \end{aligned} \tag{1.1}$$

These representations illustrate the minimal required information to define a SimpleAttributeOperand, since the IndexRange of SimpleAttributeOperands is optional, the QueryLanguage defines the BaseEventType as default value for the TypeDefinitionId and the attribute value is set as default value for the AttributeId. In consequence, these two statements create identical SimpleAttributeOperands:

1 : PATH "/Message"

2 : TYPEID i=2041 PATH "/Message" ATTRIBUTE 13

The following table clearifies the SimpleAttributeOperands defined in Equation 1.1

Table 1.1: SelectClauses

TypeDefinitionNode	IncludesSubtypes	RelativePath	Attribute	IndexRange
i=2041 (BaseEventType)	True	/Message	13 (Value)	N/A
i=2041 (BaseEventType)	True	/0:Severity	13 (Value)	N/A
i=2041 (BaseEventType)	True	/EventType	13 (Value)	N/A

1.3.2 Complex Operators

Since all Operators used for the tutorial 1.3.3 only contain a maximum of two operands, the following shortly illustrates the usage of the the InList and the Between operators which allow more than two operands:

WHERE

OR($"ref_1", $"ref_3")

FOR

$"ref_1" := PATH "/Severity" BETWEEN [$"ref_2", UINT32 18]

$"ref_2" := UINT32 10

$"ref_3" := PATH "/Severity" INLIST [$"ref_2", UINT32 120, UINT32 140]

The ContentFilter statement limits the Severity to be either between 10 and 18 ($"ref_1") or its value has to be equal to 10, 120 or 140 ($"ref_3"). Table 1.2 should clarify the exact structure of this ContentFilter, where Element represents the index of an operator within the ContentFilterElementArray, Operator the OperatorType, Operand illustrates each operand within the operator's operand array and Reference shows either the internally generated reference, or the reference set by the user. Since the query language allows to set references for operands, some elements listed in the table do not have Element and Operator values. Depending on the number of operands for a concrete operator, Operand cells can also be empty. The InList operator has an undetermined number of operands, so that the number of operands listed in such a table depends on the size of the declared list in the operator's statement.

Table 1.2: **InList and Between Operators**

Element	**Operator**	**Operand[0]**	**Operand[1]**	**Operand[2]**	**Operand[3]**	**Reference**
0	OR	ElementOperand = 1	ElementOperand = 2			operator_reference_0
1	BETWEEN	SimpleAttributeOperand = 'NodeId: i=2041, BrowsePath: "/Severity", Attribute: Value, IndexRange: N/A'	Reference = 'ref_2'	LiteralOperand = '18'		ref_1
		LiteralOperand = '10'				ref_2
2	INLIST	SimpleAttributeOperand = 'NodeId: i=2041, BrowsePath: "/Severity", Attribute: Value, IndexRange: N/A'	Reference = 'ref_2'	LiteralOperand = '120'	LiteralOperand = '140'	ref_3

1.3.3 open62541 EventFilter Tutorial

The open62541 EventFilter tutorial gives an extensive introduction into the OPC UA EventFilter and how to implement them manually with the open62541 SDK. Since plenty of coding effort is required to create them, the open62541 query languge enables an easy alternative with reduced effort. Each of the following cases illustrates a possible query to create an EventFilter for the corresponding case from the tutorial. However, the usage of default values for SimpleAttributeOperands, multiple possibilities to declare operators and the referencing of operands and operators facilitates different queries to create identical EventFilter.

Each Case is structured as followed. First, the query statement is represented, afterwards, a figure depicts the structure of this EventFilter and lastly, the ContentFilterElementArray is clarified in a table which lists each operator and operand of the ContentFilter. Since all cases have an equal SelectClauses, the SELECT statement has been illustrated in section 1.3.1.

Case 0

The query shown below creates an EventFilter that collects the values of the Serverity, the Message and the EventType, when the EventType's NodeId corresponds to either $i = 5000$ or $ns = 1; i = 5001$.

SELECT

PATH "/Message", PATH "/0:Severity", PATH "/EventType"

WHERE

OR($"ref_1", $"ref_2")

FOR

$"ref_2" := OFTYPE $ns = 1; i = 5003$

$"ref_1" := OFTYPE $i = 3035$

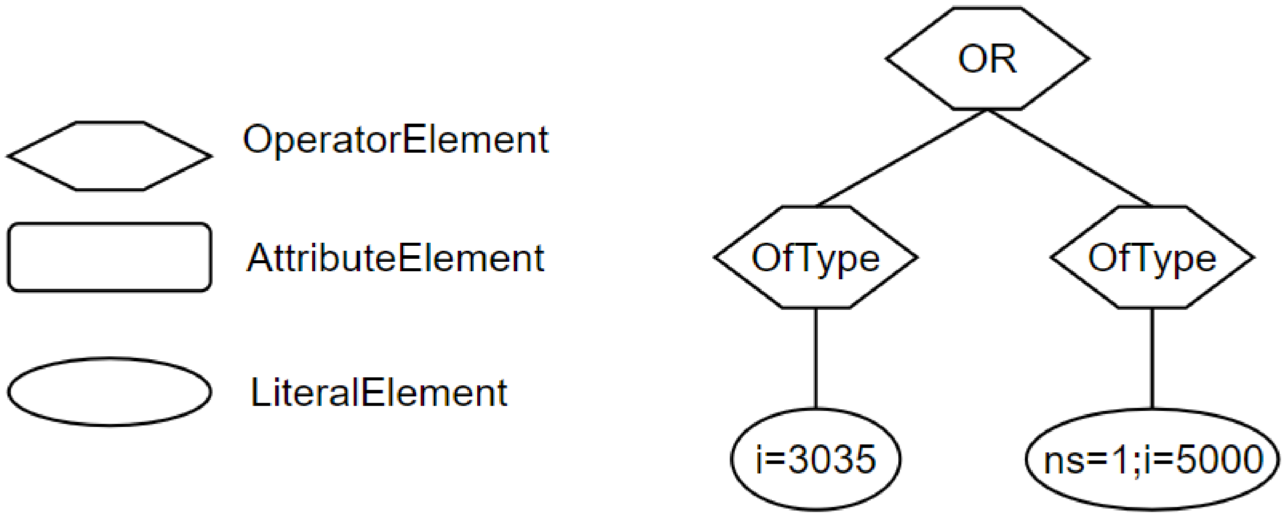

Table 1.3: **Case 0**

Element	Operator	Operand[0]	Operand[1]	Reference
0	OR	ElementOperand = 1	ElementOperand = 2	operator_reference_0
1	OFTYPE	LiteralOperand = '$i = 3035$'		ref_1
2	OFTYPE	LiteralOperand = '$ns = 1; i = 5000$'		ref_2

Case 1

The query shown below creates an EventFilter that collects the values of the Serverity, the Message and the EventType, when the EventType's NodeId corresponds to $ns = 1; i = 5001$.

SELECT

PATH "/Message", PATH "/0:Severity", PATH "/EventType"

WHERE

OFTYPE $ns = 1; i = 5001$

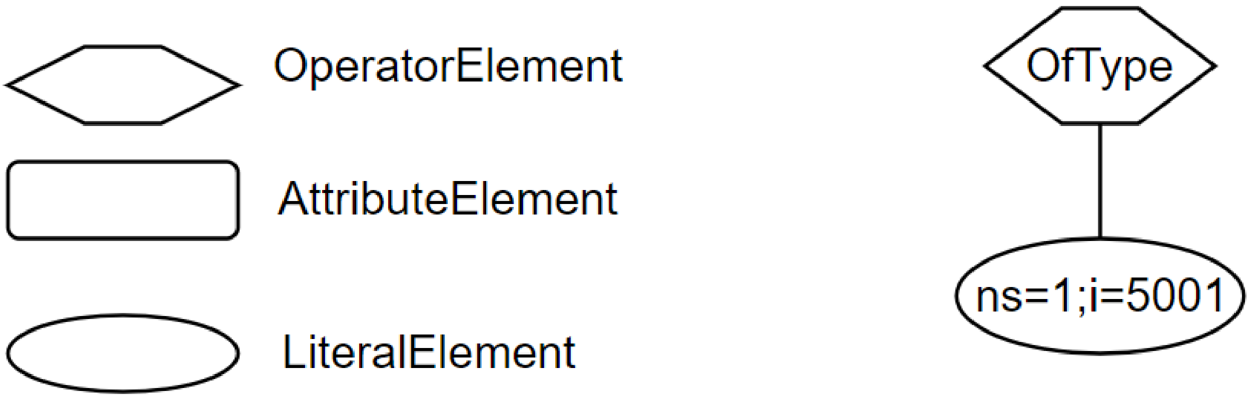

Table 1.4: **Case 1**

Element	**Operator**	**Operand[0]**	**Operand[1]**	**Reference**
0	OFTYPE	LiteralOperand = '$ns = 1; i = 5001$'		operand_reference_0

Case 2

The query shown below creates an EventFilter that collects the values of the Serverity, the Message and the EventType, when the EventType's NodeId corresponds to either $ns = 1; i = 5000$, $ns = 1; i = 5001$, $ns = 1; i = 5002$, $ns = 1; i = 5003$, $ns = 1; i = 5004$, $i = 3035$.

SELECT

PATH "/Message", PATH "/0:Severity", PATH "/EventType"

WHERE

OR($"first branch", $"second branch")

FOR

$"first branch" := OR($"third branch", $"fourth branch")

$"second branch" := OR($1, $2)

$"third branch" := OR($3, $4)

$"fourth branch" := OR($5, $6)

$1 := OFTYPE $ns = 1; i = 5000$

$2 := OFTYPE $7

$3 := OFTYPE $ns = 1; i = 5002$

$4 := OFTYPE $ns = 1; i = 5003$

$5 := OFTYPE $8

$6 := OFTYPE $9

$7 := PATH "/0:SampleDeviceFailureEventType" ATTRIBUTE 1

$\$8 := ns = 1; i = 5004$

$9 := NODEID $i = 3035$

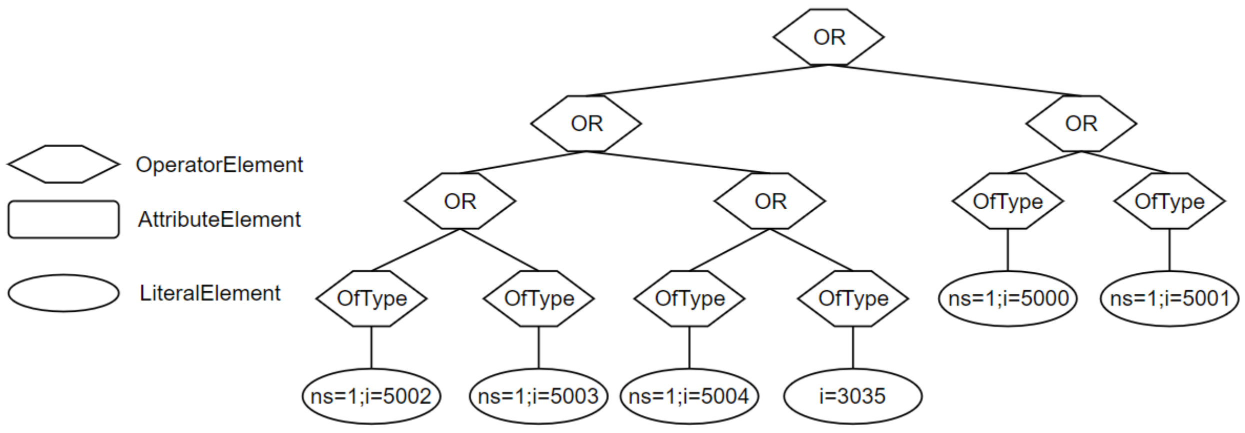

Table 1.5: **Case 2**

Element	**Operator**	**Operand[0]**	**Operand[1]**	**Reference**
0	OR	ElementOperand = '1'	ElementOperand = '2'	operator_reference_0
1	OR	ElementOperand = '3'	ElementOperand = '4'	first branch
2	OR	ElementOperand = '5'	ElementOperand = '6'	second branch
3	OR	ElementOperand = '7'	ElementOperand = '8'	third branch
4	OR	ElementOperand = '9'	ElementOperand = '10'	fourth branch
5	OFTYPE	LiteralOperand = '$ns = 1; i = 5000$'		1
6	OFTYPE	Reference = '7'		2
7	OFTYPE	LiteralOperand = '$ns = 1; i = 5002$'		3
8	OFTYPE	LiteralOperand = '$ns = 1; i = 5003$'		4
9	OFTYPE	Reference = '8'		5
10	OFTYPE	Reference = '9'		6
		SimpleAttributeOperand = 'NodeId: i=2041, BrowsePath: "/0:SampleDeviceFailureEventType", Attribute: NodeId, IndexRange: N/A'		7
		LiteralOperand = '$ns = 1; i = 5004$'		8
		LiteralOperand = '$i = 3035$'		9

Case 3

The query shown below creates an EventFilter that collects the values of the Serverity, the Message and the EventType, when the EventType's NodeId corresponds to $ns = 1; i = 5001$, the literal value 99 is equal to the literal value 99 and the event's severity value is greather than 99.

SELECT

PATH "/Message", PATH "/0:Severity", PATH "/EventType"

WHERE

AND(OFTYPE ns=1;i=5001, $3)

FOR

$1 := 99 "==" 99

$2 := TYPEID i=5000 PATH "/Severity" GT 99

$3 := AND ($1, $2)

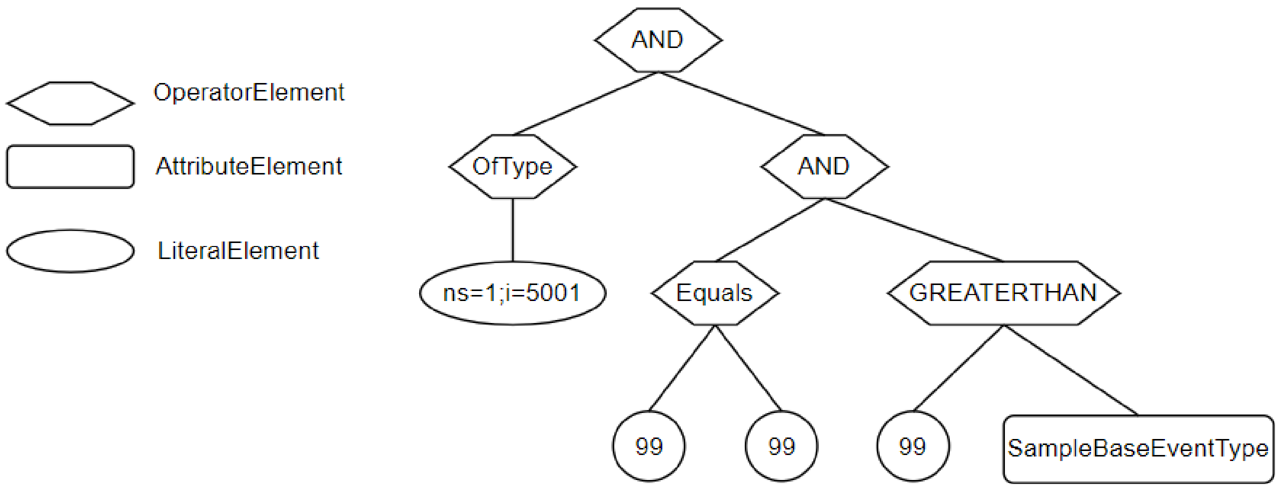

Table 1.6: **Case 3**

Element	**Operator**	**Operand[0]**	**Operand[1]**	**Reference**
0	AND	ElementOperand = '1'	ElementOperand = '2'	operator_reference_0
1	OFTYPE	LiteralOperand = '$ns = 1; i = 5001$'		operand_reference_0
2	AND	ElementOperand = '3'	ElementOperand = '4'	3
3	EQUALS	LiteralOperand = '99'	LiteralOperand = '99'	1
4	GREATERTHAN	LiteralOperand = '99'	SimpleAttributeOperand = 'NodeId: i=5000, BrowsePath: "/Severity", Attribute: Value, IndexRange: N/A'	2

Case 4

The query shown below creates an EventFilter that collects the values of the Serverity, the Message and the EventType, when the EventType's NodeId corresponds to $ns = 1; i = 5000$ and the event's severity value is greather than 99.

SELECT

PATH "/Message", PATH "/0:Severity", PATH "/EventType"

WHERE

AND((OFTYPE ns=1;i=5000), TYPEID i=5000 PATH "/Severity" GREATERTHAN 99)

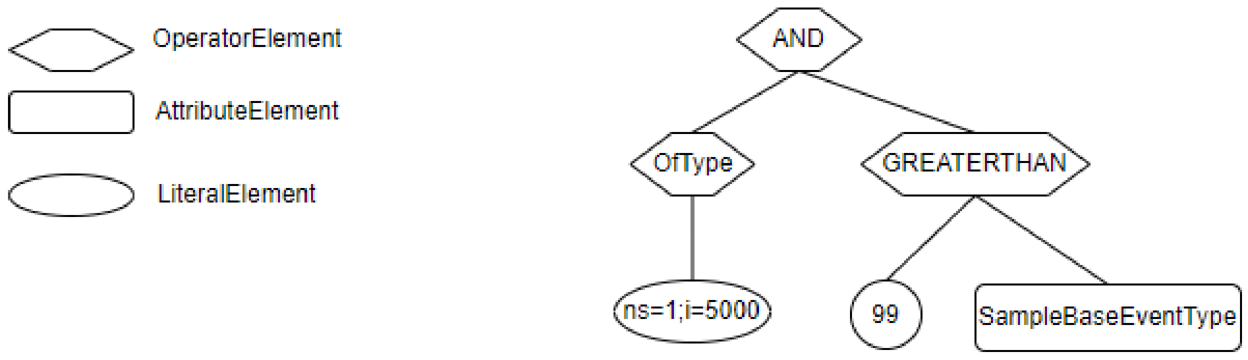

Table 1.7: **Case 4**

Element	**Operator**	**Operand[0]**	**Operand[1]**	**Reference**
0	AND	ElementOperand = ’1’	ElementOperand = ’2’	operator_reference_0
1	OFTYPE	LiteralOperand = ’$ns = 1; i = 5000$’		operand_reference_0
2	GREATERTHAN	LiteralOperand = ’99’	SimpleAttributeOperand = ’NodeId: i=5001, BrowsePath: “/Severity”, Attribute: Value, IndexRange: N/A’	operand_reference_1