ETSI TS 132 161 V18.0.0 (2024-05)

5G;
Management and orchestration;
JSON expressions (Jex)
(3GPP TS 32.161 version 18.0.0 Release 18)

Reference
RTS/TSGS-0532161vi00

Keywords
5G

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Legal Notice

This Technical Specification (TS) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities. These shall be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between 3GPP and ETSI identities can be found under https://webapp.etsi.org/key/queryform.asp.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

In the present document, modal verbs have the following meanings:

shall indicates a mandatory requirement to do something

shall not indicates an interdiction (prohibition) to do something

The constructions "shall" and "shall not" are confined to the context of normative provisions, and do not appear in Technical Reports.

The constructions "must" and "must not" are not used as substitutes for "shall" and "shall not". Their use is avoided insofar as possible, and they are not used in a normative context except in a direct citation from an external, referenced, non-3GPP document, or so as to maintain continuity of style when extending or modifying the provisions of such a referenced document.

should indicates a recommendation to do something

should not indicates a recommendation not to do something

may indicates permission to do something

need not indicates permission not to do something

The construction "may not" is ambiguous and is not used in normative elements. The unambiguous constructions "might not" or "shall not" are used instead, depending upon the meaning intended.

can indicates that something is possible

cannot indicates that something is impossible

The constructions "can" and "cannot" are not substitutes for "may" and "need not".

will indicates that something is certain or expected to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

will not indicates that something is certain or expected not to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

might indicates a likelihood that something will happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

might not indicates a likelihood that something will not happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

In addition:

is (or any other verb in the indicative mood) indicates a statement of fact

is not (or any other negative verb in the indicative mood) indicates a statement of fact

The constructions "is" and "is not" do not indicate requirements.

Introduction

Information can be represented in a structured way using markup languages. Well-known and widely used markup languages are for example XML and JSON.

It is often required to identify distinct portions in XML or JSON documents. For XML, XPath has been designed for that purpose. XPath is very powerful and includes capabilities for conditional node selection with predicates. XPath expressions can select one or more portions of an XML document.

JSON Pointer serves a similar purpose. However, its capabilities are limited compared to XPath. For example, JSON Pointer expressions can identify only a specific node or subtree of a JSON document and not multiple nodes or subtrees. Furthermore, conditions are not supported in the information selection process.

This calls for a notation applicable to JSON documents with more advanced features than JSON Pointer. This notation is called Jex (**J**SON **ex**pressions). It is inspired by and based on XPath.

Even though XPath was originally designed to select one or more nodes of an XML document, XPath expressions operate on a conceptual data model, the XPath data model. A mapping from the XML Information Set to the XPath data model is provided in Annex B of XPath 1.0 [2].

The main purpose of the Jex specification is to provide a mapping from a JSON document to the XPath data model. With this in place XPath expressions are (indirectly) applicable to JSON.

The present document will also introduce a few profiles for XPath. These profiles are designed to provide the functionality required for network and service management.

Clause 4 provides a short review of the XPath data model. Clause 6 defines the mapping of a JSON document to the XPath data model, and clause 7 introduces a few Jex profiles. Annex A demonstrates to use of Jex for network management tasks.

Readers should be familiar with XPath 1.0 [2] and JSON (IETF RFC 8259 [6]).

1 Scope

The present TS introduces JSON expressions (Jex).

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] W3C® Recommendation 16 November 1999: "XML Path Language (XPath) Version 1.0", (https://www.w3.org/TR/xpath-10/).

[3] W3C® Recommendation 14 December 2010: "XML Path Language (XPath) 2.0", (Link errors corrected 3 January 2011; Status updated October 2016), (https://www.w3.org/TR/xpath20/).

[4] W3C® Recommendation 21 March 2017: "XML Path Language (XPath) 3.1", (https://www.w3.org/TR/xpath-31/).

[5] W3C® Recommendation 21 March 2017: "XQuery and XPath Data Model 3.1", (https://www.w3.org/TR/xpath-datamodel-31/).

[6] IETF RFC 8259: "The JavaScript Object Notation (JSON) Data Interchange Format".

[7] W3C® Recommendation: "World Wide Web Consortium. Extensible Markup Language (XML) 1.0". (http://www.w3.org/TR/1998/REC-xml-19980210).

3 Definitions of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in 3GPP TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in 3GPP TR 21.905 [1].

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1].

Jex JSON Expression

4 XPath data model

The XPath data model is described in clause 5 of W3C Xpath1.0 specification [2]. It is a conceptual model without formal notation.

The model consists of nodes with relationships between them. There are seven types of nodes defined: root node, element node, text node, attribute node, namespace node, processing instruction node, comment node.

Note that the data model for XPath 2.0 [3] and XPath 3.1 [4] is described in XQuery and in XPath Data Model 3.1 [5]. This model is not used in the present document.

5 JSON restrictions

5.1 Supported JSON documents

A JSON document (JSON text) is a serialized JSON value (clause 2, IETF RFC 8259 [6]). A JSON value is a JSON object, a JSON array, a number, a string or any of the three literal names true, false or null.

A Jex expressions can be applied only against documents containing a single JSON object. All other values or any combination of values (at the top level) are not supported.

The following example document is valid. It contains a single JSON object.

```
{
  "a": 1,
  "b": 2
}
```

The next document is not supported for use with Jex expressions, though it is a valid JSON document. It contains a JSON array at the top level.

```
[
  {
    "a": 1,
    "b": 2
  }
]
```

5.2 Supported JSON arrays

A JSON array consists of an ordered list of array items. Each array item can be a scalar value, a JSON object, or a JSON array. According to clause 5 of RFC 8259 [5] there is no requirement that the values in an array are of the same type.

Jex supports only arrays with the following properties:

- The array items of an array are all of the same type.
- Array items can be only scalars or JSON objects, but not JSON arrays.

6 Mapping of JSON to the XPath data model

6.1 Mapping of JSON documents

A JSON document is mapped to the (conceptual) root node. The root node has no name.

6.2 Mapping of scalar values

A scalar value in JSON is a string, a number, or one of the tree literal names true, false or null. These values are mapped to text nodes. Text nodes in the XPath 1.0 data model have no further qualification with a data type. However, a Jex processor shall store the original data type used in the JSON document.

Note that JSON strings are enclosed by quotation marks, JSON numbers are not enclosed by quotation marks, and the three tree literal names true, false or null are not enclosed by quotation marks either. In other words, it is possible to deduct the type of the scalar value in a JSON document by inspecting the value itself.

6.3 Mapping of name/value pairs

6.3.1 Case: The value is a scalar

The name of the name/value pair is mapped to an element node. The name of the element node is equal to the name of the name/value pair.

The value of the name/value pair is mapped to a text node as described in clause 9.3.3.

The text node coming from the value of a mapped name/value pair is the child of the element node coming from the name of the mapped name/value pair. Vice versa, the element node coming from the name of the mapped name/value pair is the parent of the text node coming from the value of the mapped name/value pair.

Examples:

The data type of the scalar value is number.

`"a": 1`	`<a>1</a>`

The data type of the scalar value is string.

`"a": "string"`	`<a>string</a>`

The data type of the scalar value is the literal name null.

`"a": null`	`<a>null</a>`

The data type of the scalar value is the literal name true.

`"a": true`	`<a>true</a>`

The data type of the scalar value is the literal name false.

`"a": false`	`<a>false</a>`

6.3.2 Case: The value is a JSON object

The name of the name/value pair is mapped to an element node. The name of the element node is equal to the name of the name/value pair.

The value of the name/value pair consists of an unordered list of name/value pairs. Each name of these name/value pairs is mapped to an element node. No order can be assumed for these element nodes.

The element nodes coming from the value of the mapped name/value pair are children of the element node coming from the name of the mapped name/value pair. Vice versa, the element node coming from the name of the mapped name/value pair is the parent of the element nodes coming from the value.

Example:

`"a": {`
`  "b": 1,`
`  "c": 2`
`}`
	`<a>`
	`  <b>1</b>`
	`  <c>2</c>`
	`</a>`

or

"a": {
  "b": 1,
  "c": 2
}
	<a>
	  <c>2</c>
	  1
	

"a": {
  "b": 1,
  "c": {
  "d": 2,
  "e": 3
  }
}
	<a>
	  1
	  <c>
	  <d>2</d>
	  <e>3</e>
	  </c>
	

6.3.3 Case: The value is a JSON array

The name of the name/value pair is mapped to a specific number of element nodes. The number of element nodes is equal to the number of array items. The names of these element nodes are all identical and equal to the name of the name/value pair.

The order of element nodes is the same as the order of the array items in the corresponding JSON.

The element nodes coming from the value of the mapped name/value pair are children of the element node coming from the name of the mapped name/value pair. Vice versa, the element node coming from the name of the name/value pair is the parent of the element nodes coming from the array items of the JSON array.

Example:

"a": [
  1,
  2,
  3
]
	<a>1
	<a>2
	<a>3

"a": [
  {"b": 1,
  "c": 2
  },
  {"b": 3,
  "c": 4
  },
  {"b": 5,
  "c": 6
  },
]
	<a>
	  1
	  <c>2</c>
	
	<a>
	  3
	  <c>4</c>
	
	<a>
	  5
	  <c>6</c>
	

6.3.4 XPath data model concepts required by JSON

A JSON document is mapped to root nodes, element nodes and text nodes. Attribute nodes, namespace nodes, processing instruction nodes and comment nodes have no equivalent in JSON.

The concept of document order is applicable only for element nodes coming from JSON arrays.

The concept of variables is not used in Jex.

The root node in the XPath 1.0 data model may have only one element node as child (the document element). This restriction is relaxed in Jex. The root node may have any number of element nodes as children.

Example:

{
  "a": 1,
  "b": 2,
  "c": 3
	<a>1
	2
	<c>3</c>

```
}
```

7 Jex expressions

7.1 Introduction

Jex uses the same syntax, the same concepts and the same definitions as XPath. Jex expressions are a subset of XPath expressions. All subsets support only the abbreviated syntax.

A Jex expression is applied to an input JSON document. The output of a Jex expression is always a node set, or one of the boolean values true and false.

Different XPath subsets are defined in the following clauses A subset is also called Jex profile.

7.2 Basics

7.2.1 Evaluation context

Jex expressions are evaluated in a context, that is a subset of the XPath evaluation context. The Jex context includes

- a node (the context node)
- a pair of non-zero positive integers (the context position and the context size)
- a function library

The initial context node of a Jex expression is specified where the Jex pression is used. This initial context node is often referred to as base object.

Note that context position and context size work only for element nodes coming from JSON arrays.

7.2.2 Scalar values

7.2.2.1 String

The representation of a string is as defined in IETF RFC 8259 [6], clause 7. A string is always surrounded by quotation marks.

```
String ::= '"' StringChar '"'
StringChar ::= #'[^"\/\b\f\n\r\t]*'
```

7.2.2.2 Number

The representation of a number is as defined in IETF RFC 8259 [6], clause 6.

```
Number        ::= ( Minus )? NonNegativeInteger ( Fraction )? ( Exponent )?
DecimalPoint ::= '.'
Zero  ::= '0'
Digit0-9 ::= #'[0-9]+'
Digit1-9 ::= #'[1-9]+'
e ::= 'e' | 'E'
Exponent ::= e ( Minus | Plus )? Digit0-9+
Fraction ::= DecimalPoint Digit0-9+
NonNegativeInteger ::= Zero | ( Digit1-9 Digit0-9* )
Minus ::= '-'
Plus  ::= '+'
```

7.2.2.3 Literal strings

The three string literals are

```
true ::= 'true'
false ::= 'false'
null ::= 'null'
```

7.2.3 Error handling

A syntax error in the Jex expression results in no output or the bloolean value false

A data type mismatch in comparisons returns always false.

A Jex processor may evaluate Jex expressions based on the schema definition of the object tree the Jex expression is applied to. In this case the Jex expressions will never evaluate to true or a non-empty node set, for example because of a misspelled data node name, the Jex processor may raise an error. Details of error detection and how an error is notified is out of scope of the present document.

7.2.4 White space handling

White spaces are not allowed in a Jex expression. An exception are "and" and "or" expressions where exactly one white space character shall be present before and after the operator.

7.3 The location path

A location path selects zero or more data nodes in an object tree. The location path is either an absolute location path or a relative location path. An absolute location path starts at the root node. A relative location path starts at the context node.

```
LocationPath ::= AbsoluteLocationPath | RelativeLocationPath
```

An absolute location path consists of "/", optionally followed by a relative location path. A "/" by itself selects the root node.

```
AbsoluteLocationPath ::= '/' RelativeLocationPath?
```

A relative location path consists of one or more axis steps. Only the child axis is supported in Jex. The location step contains a data node name test and an optional predicate. The asterisk "*" is supported and selects all element children of the context node.

```
RelativeLocationPath ::= ChildAxisStep ('/' ChildAxisStep)*
ChildAxisStep ::= DataNodeNameTest Predicate? | AbbreviatedStep
DataNodeNameTest ::= DataNodeName | '*'
```

The predicate is an expression encapsulated in rectangular brackets. The predicate expression shall evaluate to true or false. The child node is selected only when the predicate evaluates to true. The capabilities of the predicate differ for the different Jex profiles. The predicate expressions are defined where the profiles are defined.

```
Predicate ::= '[' PredicateExpr ']'
PredicateExpr ::= JexBasicPredicateExpr | JexAdvancedPredicateExpr
```

The abbreviated step selects the current node.

```
AbbreviatedStep ::= '.'
```

The "DataNodeName" is either a class name, the string "attributes", an attribute name, an attribute field name, or a notification parameter. The characters delineating the components of a Jex expression are not allowed.

```
DataNodeName ::= #'[^"[\]=!<>\n ]*'
```

Note that depending on the context where the Jex expression is used more characters may be excluded.

Examples of an absolute location path without predicates:

```
/SubNetwork/ManagedElement/attributes
/SubNetwork/ManagedElement/attributes/userLabel
```

Examples of an absolute location path with predicates:

```
/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes
/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes/userLabel
```

7.4 Jex basic for node selection

An expression in Jex basic returns a set of data nodes. The output node set may be empty. The data nodes that can be selected are managed object instances, attributes, attribute fields and attribute elements of multi-valued attributes. Conditional data node selection with predicates is limited.

Each Jex basic expression is an absolute location path.

```
JexBasicExpr ::= AbsoluteLocationPath
```

Predicates are used in Jex basic only for selecting

- element nodes representing managed object instances based on the value of their naming attribute "id".
- array items representing attribute elements (of multi-valued attributes) based on their positional index. The first element has the index "0".

Other conditions are not supported in predicates.

```
JexBasicPredicateExpr ::= MoiSelectorExpr | AttributeElementSelector
MoiSelectorExpr ::= 'id=' String
AttributeElementSelector ::= NonNegativeInteger
```

The function library in Jex Basic is empty.

Examples:

The following Jex expression selects all attributes of the "SubNetwork" whose "id" is SN1, or the complete managed object, depending on the context.

```
/SubNetwork[id="SN1"]/attributes
```

In the next examples the Jex expressions select one attribute of a specific manged object.

```
/SubNetwork[id="SN1"]/attributes/userLabel
/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes/vendorName
```

An example for selecting an attribute field may look as follows.

```
/SubNetwork[id="SN1"]/attributes/plmnId/mcc
```

All attributes of an object instance can be selected with the wildcard "*".

```
/SubNetwork[id="SN1"]/attributes/*
```

The following expression selects the first attribute element of a multi-valued attribute.

```
/SubNetwork[id="SN1"]/ThresholdMonitor[id="TM1"]/attributes/thresholdLevels[0]
```

The following example shows how all "ManagedElement" instances, that are childs of the "SubNetwork with the "id" equal to SN1, are selected by the second location step with a name test. The following location steps select then the "vendorName" attribute of the previously selected "ManagedElement" instances.

```
/SubNetwork[id="SN1"]/ManagedElement/attributes/vendorName
```

Note that the EBNF allows also JEX expressions that do not make sense and result in empty node output sets in most cases.

```
/SubNetwork[id="SN1"]/attributes[id="A1"]
/SubNetwork[id="SN1"]/attributes/userLabel[2]
```

```
/SubNetwork[2]/attributes/plmnId/mcc
```

7.5 Jex advanced for node selection

An expression in Jex advanced returns a set of data nodes. The output node set may be empty. The data nodes that can be selected are managed object instances, attributes, attribute fields and attribute elements of multi-valued attributes. Conditional data node selection with predicates is much more powerful than in Jex basic.

An expression in Jex advanced is a union of absolute path expressions. The path expression is more flexible than the location path and allows to select element nodes with more than one data node name in a single step.

```
JexAdvancedExpr ::= AbsolutePathExpr | (AbsolutePathExpr)*
AbsolutePathExpr ::= ('/' RelativePathExpr?)
RelativePathExpr ::= StepExpr ('/' StepExpr)*
StepExpr ::= ChildFilterExpr | ChildAxisStep
ChildFilterExpr ::= '(' UnionExpr ')'
UnionExpr ::= LocationPath ('|' LocationPath)*
LocationPath ::= AbsoluteLocationPath | RelativeLocationPath
```

Examples:

```
/SubNetwork/(ManagedElement|PerfMetricJob)/attributes
/SubNetwork/(ManagedElement/XyzFunction|ThresholdMonitor)/attributes
/SubNetwork/ManagedElement/attributes/(userLabel|vendorName)
/SubNetwork/attributes/userLabel|/SubNetwork/ManagedElement/attributes/userLabel
```

Jex Advanced also extends Jex Basic with more powerful predicates for selecting data nodes by supporting comparison expressions that can be combined using "or" and "and".

```
Predicate                ::= '[' JexAdvancedPredicateExpr ']'
JexAdvancedPredicateExpr ::= OrExpr | AttributeElementSelector
OrExpr                   ::= AndExpr (' or ' AndExpr)*
AndExpr                  ::= AndOperandExpr (' and ' AndOperandExpr)*
AndOperandExpr           ::= ComparisionExpr| LocationPath | FunctionCall | '(' OrExpr ')' | 'not'
'(' OrExpr ')'
ComparisionExpr          ::= EqualityExpr | RelationalExpr
AttributeElementSelector ::= NonNegativeInteger
```

The "and" operator has a higher precedence than the "or" operator. An "and" expression, an "or" expression or any combination of an "and" and an "or" expressions can be enclosed in parentheses, which allows to control the precedence of the "and" and "or" operators.

Equality and relational expressions have on the left side of the operator an absolute location path or a relative location path. On the right side of the operator equality expressions have a string, a number or one of the three literals true, false, or null. Relational expressions have on the right side of the operator a number.

```
EqualityExpr   ::= LocationPath ('=' | '!=') (String
                                              | Number
                                              | true | false | null)
RelationalExpr ::= LocationPath ('<' | '>' | '<=' | '>=' ) Number
```

The data type of the value on the right side of the operator shall have the same data type as the value produced by the location path on the left side of the operator. If the values do not have the same data type the equality or relational expression shall evaluate to "false".

The function library in Jex Advanced features the XPath string function "contains" defined in clause 4.2 of the W3C XPath1.0 specification [2], and the boolean function "not" defined in clause 4.3 of the W3C XPath1.0 specification [2].

The "not" function returns true if its argument is false, and false otherwise. Its argument is an "or" expression.

Example:

```
/SubNetwork/ManagedElement/attributes[vendorName="Company XY" and location="TV Tower"]
/SubNetwork/ManagedElement/attributes[vendorName="Company XY" and not(location="TV Tower")]
```

The "contains" function returns true if the first argument string contains the second argument string, and otherwise returns false. It can be used only in predicates as follows.

```
FunctionCall ::= FunctionName'('LocationPath','Argument')'
```

```
FunctionName ::= "contains"
Argument     ::= String
```

Examples:

In the first example the specified "ManagedElement" instance is selected only when the "vendorName" attribute has the value "Company XY".

```
/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes[vendorName="Company XY"]
```

/SubNetwork[id="SN1"]/ManagedElement[id="ME1" and attributes/vendorName="Company XY"]

Instead of the instance only one attribute can be selected.

```
/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes[vendorName="Company XY"]/userLabel
```

The Jex expression in the next example selects all "ManagedElements" from the vendor "Company XY".

```
/SubNetwork[id="SN1"]/ManagedElement/attributes[vendorName="Company XY"]
```

The following example selects the threshold level identified by the "level" 3.

```
/SubNetwork[id="SN1"]/ThresholdMonitor[id="TM1"]/attributes/ThresholdLevels[level=3]
```

The location paths in the predicates in the examples above are relative location paths with a single location step. Multiple location steps are also possible in a predicate.

```
/SubNetwork[id="SN1"]/attributes[plmnId/mnc=789]
```

The location path in a predicate can also be an absolute location path. This allows to test on conditions prevailing somewhere else in the object tree.

```
/SubNetwork[id="SN1"]/ManagedElement/attributes[/SubNetwork[id="SN1"]/PerfMetricJob[id="PMJ1"]\
/attributes/attrA=1]
```

The following example shows how multiple attributes can be selected using a sequence expression.

```
/SubNetwork[id="SN1"]/attributes/(userLabel|userDefinedNetworkType)
```

Sequence expressions can also be used to select objects of different classes.

```
/SubNetwork[id="SN1"]/(ThresholdMonitor|PerfMetricJob)/attributes
```

Multiple conditions can appear in a predicate.

```
/SubNetwork[id="SN1"]/attributes[userLabel="Berlin NW" and userDefinedNetworkType="5G"]
```

The following expression selects all "ManagedElement" objects whose "location" value contains the string "tower".

```
/SubNetwork/ManagedElement/attributes[contains(location,"tower")]
```

The next expression returns the "location" only if its value contains the string "tower".

```
/SubNetwork/attributes/location[contains(.,"tower")]
```

7.6 Jex conditions for condition evaluation

A Jex conditions expression evaluates to "true" or "false". It is equal to the predicate expression in Jex advanced.

```
JexConditionsExpr ::= JexAdvancedPredicateExpr
```

Jex conditions is used for evaluating conditions in an object tree or some other JSON document. Depending on the outcome certain actions may be triggered. For example, Jex conditions can be used for notification filtering, or starting and stopping collection of additional performance metrics when some basic permanently monitored performance metrics are above a certain threshold.

Annex A (normative): Jex grammar

A.1 EBNF

The grammar of Jex is specified using the Extended Backus-Naur Form (EBNF) notation defined in XML 1.0 [7], with the following addition:.

- Regular expressions can be specified using `#'regEx'`

A.2 EBNF for Jex basic

This appendix specifies the normative version of the complete EBNF for Jex basic.

```
ManyPaths ::= (JexBasicExpr "\n" | "\n" | ("&&" #'[^\n]*' "\n")) *

JexBasicExpr ::= AbsoluteLocationPath
AbsoluteLocationPath ::= '/' RelativeLocationPath
RelativeLocationPath ::= ChildAxisStep ('/' ChildAxisStep)*
ChildAxisStep ::= DataNodeNameTest Predicate? | AbbreviatedStep
DataNodeNameTest ::= DataNodeName | '*'
AbbreviatedStep ::= '.'
Predicate ::= '[' JexBasicPredicateExpr ']'
JexBasicPredicateExpr ::= MoiSelectorExpr | AttributeElementSelector
MoiSelectorExpr ::= 'id=' String
AttributeElementSelector ::= NonNegativeInteger

NonNegativeInteger ::= #'[0-9]+'
DataNodeName ::= #'[^"[\]=!<>\n]*'

String ::= '"' StringChar '"'
StringChar ::= #'[^"\/\b\f\n\r\t]*'
```

A.3 EBNF for Jex advanced

This appendix specifies the normative version of the complete EBNF for Jex advanced.

```
ManyPaths ::= (JexAdvancedExpr "\n" | "\n" | ("&&" #'[^\n]*' "\n")) *
JexAdvancedExpr ::= AbsolutePathExpr
AbsolutePathExpr ::= ('/' RelativePathExpr?)
RelativePathExpr ::= StepExpr ('/' StepExpr)*
StepExpr ::= ChildFilterExpr | ChildAxisStep
ChildFilterExpr ::= '(' UnionExpr ')'
UnionExpr ::= LocationPath ('|' LocationPath)*

LocationPath ::= AbsoluteLocationPath | RelativeLocationPath
AbsoluteLocationPath ::= '/' RelativeLocationPath
RelativeLocationPath ::= ChildAxisStep ('/' ChildAxisStep)*
ChildAxisStep ::= DataNodeNameTest Predicate? | AbbreviatedStep

DataNodeNameTest ::= '*' | DataNodeName
Predicate        ::= '[' JexAdvancedPredicateExpr ']'
AbbreviatedStep  ::= '.'

JexAdvancedPredicateExpr ::= OrExpr | AttributeElementSelector
AttributeElementSelector ::= NonNegativeInteger
OrExpr                   ::= AndExpr (' or ' AndExpr)*
AndExpr                  ::= AndOperandExpr (' and ' AndOperandExpr)*
AndOperandExpr  ::= ComparisionExpr | LocationPath | FunctionCall | '(' OrExpr ')' | 'not' '('
OrExpr ')'
ComparisionExpr ::= EqualityExpr | RelationalExpr

EqualityExpr   ::= LocationPath ('=' | '!=') (String
                                              | Number
```

```
                                              | true | false | null)
RelationalExpr ::= LocationPath ('<' | '>' | '<=' | '>=' ) Number

FunctionCall ::= FunctionName'('LocationPath','Argument')'
FunctionName ::= "contains"
Argument     ::= String

DataNodeName       ::= #'[^"[\]=!<>\n(),/]+'

String       ::= '"' StringChar '"'
StringChar         ::= #'[^"]*'

Number        ::= ( Minus )? NonNegativeInteger ( Fraction )? ( Exponent )?
DecimalPoint ::= '.'
Zero  ::= '0'
Digit0-9 ::= #'[0-9]+'
Digit1-9 ::= #'[1-9]+'
e ::= 'e' | 'E'
Exponent ::= e ( Minus | Plus )? Digit0-9+
Fraction ::= DecimalPoint Digit0-9+
NonNegativeInteger ::= Zero | ( Digit1-9 Digit0-9* )
Minus ::= '-'
Plus  ::= '+'

true               ::= 'true'
false              ::= 'false'
null               ::= 'null'
```

A.4 EBNF for Jex conditions

This appendix specifies the normative version of the complete EBNF for Jex conditions.

```
ManyPaths ::= (JexConditionsExpr "\n" | "\n" | ("&&" #'[^\n]*' "\n")) *
JexConditionsExpr ::= JexAdvancedPredicateExpr

LocationPath ::= AbsoluteLocationPath | RelativeLocationPath
AbsoluteLocationPath ::= '/' RelativeLocationPath
RelativeLocationPath ::= ChildAxisStep ('/' ChildAxisStep)*
ChildAxisStep ::= DataNodeNameTest Predicate? | AbbreviatedStep

DataNodeNameTest ::= '*' | DataNodeName
Predicate        ::= '[' JexAdvancedPredicateExpr ']'
AbbreviatedStep  ::= '.'

JexAdvancedPredicateExpr ::= OrExpr | AttributeElementSelector
OrExpr         ::= AndExpr (' or ' AndExpr)*
AndExpr        ::= AndOperandExpr (' and ' AndOperandExpr)*
AndOperandExpr ::= ComparisionExpr | LocationPath | FunctionCall | '(' OrExpr ')' | 'not' '(' OrExpr
')'
ComparisionExpr ::= EqualityExpr | RelationalExpr
AttributeElementSelector ::= NonNegativeInteger

EqualityExpr   ::= LocationPath ('=' | '!=') (String
                                              | Number
                                              | true | false | null)
RelationalExpr ::= LocationPath ('<' | '>' | '<=' | '>=' ) Number

FunctionCall ::= FunctionName'('LocationPath','Argument')'
FunctionName ::= "contains"
Argument     ::= String

DataNodeName       ::= #'[^"[\]=!<>\n(),/]+'

String       ::= '"' StringChar '"'
StringChar         ::= #'[^"]*'

Number        ::= ( Minus )? NonNegativeInteger ( Fraction )? ( Exponent )?
DecimalPoint ::= '.'
Zero  ::= '0'
Digit0-9 ::= #'[0-9]+'
Digit1-9 ::= #'[1-9]+'
e ::= 'e' | 'E'
Exponent ::= e ( Minus | Plus )? Digit0-9+
Fraction ::= DecimalPoint Digit0-9+
```

```
NonNegativeInteger ::= Zero | ( Digit1-9 Digit0-9* )
Minus ::= '-'
Plus  ::= '+'

true               ::= 'true'
false              ::= 'false'
null               ::= 'null'
```

Annex B (informative): EBNF test cases

B.1 Jex basic

```
/SubNetwork
/SubNetwork/attributes
/SubNetwork[id="SN1"]
/SubNetwork[id="SN1"]/attributes
/SubNetwork[id="SN1"]/attributes/userLabel
/SubNetwork[id="SN1"]/attributes/plmnId/mcc
/SubNetwork[id="SN1"]/attributes/plmnId/*
/SubNetwork[id="SN1"]/ManagedElement/attributes/vendorName
/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes/vendorName
/SubNetwork[id="SN1"]/ThresholdMonitor[id="TM1"]/attributes/ThresholdLevels[0]
```

B.2 Jex advanced

```
/SubNetwork[id="SN1"]/ManagedElement
/SubNetwork[id="SN1"]/ManagedElement/attributes
/SubNetwork[id="SN1"]/ManagedElement/attributes/vendorName
/SubNetwork[id="SN1"]/ManagedElement[attributes/vendorName="Company XY"]
/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]
/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes
/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes/opState
/SubNetwork[id="SN1"]/ManagedElement[attributes/vendorName="Company XY"]
/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes or
/SubNetwork[id="SN2"]/ManagedElement/attributes/vendorName
/SubNetwork[id="SN2"]/(ManagedElement|ThresholdMonitor)/attributes

/SubNetwork[id="SN1"]/ThresholdMonitor[id="TM1"]/attributes/ThresholdLevels[level=3]
/SubNetwork[id="SN1"]/ThresholdMonitor[id="TM1"]/attributes/ThresholdLevels[3]

/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes/(opState|adminState)
 /SubNetwork/(ManagedElement|ThresholdMonitor)/attributes

/SubNetwork/attributes/userLabel|/SubNetwork/ManagedElement/attributes/userLabel

/SubNetwork[id="SN1"]/ManagedElement[id="ME1" and attributes/vendorName="Company XY"]
/SubNetwork[id="SN1"]/ManagedElement[id="ME1" and attributes/vendorName="Company XYZ" or
attributes/userLabel="Berlin NW 1"]/attributes/userLabel
/SubNetwork/attributes[userLabel="Berlin NW" and (plmnId/mcc=456 or plmnId/mcc=457)]

/SubNetwork/ManagedElement/attributes[contains(userLabel,"Berlin")]

/SubNetwork/ManagedElement/attributes[not(userLabel="Berlin NW 1")]
/SubNetwork/ManagedElement/attributes[not(contains(userLabel,"Berlin"))]
/SubNetwork[id="SN1"]/ThresholdMonitor[id="TM1"]/attributes/ThresholdLevels[not(level=1)]
/SubNetwork/attributes[not(userLabel="Berlin NW" and not(plmnId/mcc=456 or not(plmnId/mcc=457)))]
```

B.3 Jex conditions

```
/notificationType="notifyNewAlarm" and /perceivedSeverity="CRITICAL"
notificationType="notifyNewAlarm" and perceivedSeverity="CRITICAL"

(notificationType="notifyNewAlarm" or notificationType="notifyClearedAlarm" or
notificationType="notifyChangedAlarmGeneral" or notificationType="notifyAckStateChanged" or
notificationType="notifyComments") and perceivedSeverity="MAJOR" or
notificationType="notifyCorrelatedNotificationChanged"

alarmType="COMMUNICATIONS_ALARM" or alarmType="EQUIPMENT_ALARM"

(probableCause ="degradedSignal " or probableCause ="transmitFailure") and
contains(additionalInformation,"RSU_22") and  perceivedSeverity="MAJOR" or
perceivedSeverity="CRITICAL"
```

```
notificationType="notifyNewAlarm" and (alarmType="Communications Alarm" and
perceivedSeverity="CRITICAL")

notificationType="notifyNewAlarm" and specificProblem[contains(.,"Flood")] and
perceivedSeverity="CRITICAL"

notificationType="notifyNewAlarm" and specificProblem[contains(.,"Fire")] and
(perceivedSeverity="CRITICAL" or perceivedSeverity="MAJOR")

notificationType="notifyChangedAlarmGeneral" and contains(specificProblem,"Fire") and
(perceivedSeverity="CRITICAL" or perceivedSeverity="MAJOR")

(notificationType="notifyNewAlarm" and perceivedSeverity="CRITICAL") or
(notificationType="notifyChangedAlarmGeneral" and contains(specificProblem,"Fire") and
(perceivedSeverity="MINOR" or perceivedSeverity="MAJOR"))

(contains(href,"GNDBDUFunction") and perceivedSeverity="CRITICAL") or contains(href,"CUCPFunction")

(notificationType="notifyNewAlarm" and (perceivedSeverity="CRITICAL" or perceivedSeverity="MAJOR"))
or (notificationType="notifyChangedAlarmGeneral" and (perceivedSeverity="CRITICAL") or
notificationType="notifyClearedAlarm")

specificProblem!="CPUOverHeat" or monitoredAttributes/monitoredValue>=5.67

monitoredAttributes/attr1/field1/subfield2="4a"

notificationType="notifyFileReady" and fileInfoList/fileDataType ="TRACE"
```

Annex C (informative): Comparison of Jex with XPath 1.0

C.1 Comparison of Jex basic with XPath 1.0

The function library in Jex Basic is empty.

Only one predicate is allowed per step.

Only the following operators are supported "=".

Use of parenthesis is not supported.

Namespaces are not supported.

Only the child axis is supported.

The concept of variables is not used in Jex.

C.2 Comparison of Jex advanced with XPath 1.0

The function library in Jex Basic includes only the "contains()" and "not()" function.

Only one predicate is allowed per step.

Only the following operators are supported "=", "!=", "<", ">", "<=", ">=", "or", "and".

Namespaces are not supported.

Only the child axis is supported.

The concept of variables is not used in Jex.

When using comparison operators "=", "!=", "<", ">", ">=" or "<=" both sides result in a single scalar value: string, number, or true, false or null. If this is not the case the expression has the value false. E.g., if multiple attributes or a multivalued attribute or an MOI is selected by the left side the result will be false.

C.3 Comparison of Jex conditions with XPath 1.0

A Jex conditions expression is equal to the predicate in Jex advanced. Therefore the statements made for the predicate in Jex advanced apply here.

Annex D (informative): Example use cases

All these use cases have in common that one or more data nodes need to be identified. To these selected nodes a certain semantics is attached.

Notification subscription

1 Subscribe to all alarm notifications of one specific "ManagedElement" instance below a specific "SubNetwork" instance. Note, for subscribing to alarm notifications the Jex expression identifies a set of managed objects.

```
"dataNodeSelector": "/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes",
"notificationTyes": ["alarmNotifications"]
```

2 Subscribe to all alarm notifications of all "ManagedElement" instances below a specific "SubNetwork".

```
"dataNodeSelector": "/SubNetwork[id="SN1"]/ManagedElement/attributes",
"notificationTyes": ["alarmNotifications"]
```

3 Subscribe to all alarm notifications of the managed object tree whose root object is a specific "ManagedElement" instance.

```
"dataNodeSelector": "/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]",
"notificationTyes": ["alarmNotifications"]
```

4 Subscribe to all alarm notifications of all managed object trees whose root objects are "ManagedElement" instances below a specifc "SubNetwork" instance.

```
"dataNodeSelector": "/SubNetwork[id="SN1"]/ManagedElement",
"notificationTyes": ["alarmNotifications"]
```

5 Subscribe to all alarm notifications of "ManagedElement" instances from vendor "Company XY" below a specific "SubNetwork" instance.

```
"dataNodeSelector": "/SubNetwork[id="SN1"]/ManagedElement/attributes[vendorName="Company XY"]",
"notificationTyes": ["alarmNotifications"]
```

6 Subscribe to all alarm notifications of the object subtrees whose root objects are "ManagedElement" instances from vendor "Company XY".

```
"dataNodeSelector": "/SubNetwork[id="SN1"]/ManagedElement[attributes/vendorName="Company XY"]",
"notificationTyes": ["alarmNotifications"]
```

7 Subscribe to attribute value change notifications of a specific attribute of a specific managed object.

```
"dataNodeSelector": "/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes/opState",
"notificationTyes": ["avcNotification"]
```

8 Subscribe to attribute value change notifications of multiple specific attributes of a specific managed object.

```
"dataNodeSelector": "/SubNetwork[id="SN1"]/ManagedElement[id="ME1"]/attributes/(opState |
adminState)",
"notificationTyes": ["avcNotification"]
```

Notification filtering

Jex conditions allows to select notifications based on the values of notification parameters. Note that in a JSON document with the representation of a notification all notification parameters are on the top level just below the root node. The following example shows an alarm notification.

```
{
  "href": "example.com/SubNetwork=1",
  "notificationId": "34",
  "notificationType": "notifyNewAlarm",
  "eventTime": "2021-12-19T16:39:57-08:00",
  "systemDN": "example.com,SubNetwork=SN1,MnsAgent=MA1",

  "alarmId": "alarmId1",
  "alarmType": "QUALITY_OF_SERVICE_ALARM",
  "probableCause": "SYSTEM_RESOURCES_OVERLOAD",
  "perceivedSeverity": "CRITICAL"
}
```

The following Jex expressions are examples for notification filters that can be applied to this notification. When the Jex expression evaluates to true, the notification is forwarded, otherwise it is discarded.

```
notificationFilter: "perceivedSeverity="CRITICAL""

notificationFilter: "notificationType="notifyNewAlarm" and perceivedSeverity="CRITICAL"

notificationFilter: "alarmType="QUALITY_OF_SERVICE_ALARM" and perceivedSeverity="CRITICAL""

notificationFilter: "specificProblem[contains(.,"Flood")] and perceivedSeverity="CRITICAL""

notificationFilter: "specificProblem[contains(.,"Fire")] \
                     and (perceivedSeverity="CRITICAL" or perceivedSeverity="MAJOR")"
```

Annex E (informative): Change history

Change history							
Date	**Meeting**	**TDoc**	**CR**	**Rev**	**Cat**	**Subject/Comment**	**New version**
2023-10	SA5#151	S5-236575				Initial skeleton	0.0.0
2023-10	SA5#151	S5-236577				Rel-18 pCR 32.161 Add Jex definitions	0.1.0
2024-02	SA5#153	S5-240841				Rel-18 pCR 32.161 Improve Jex	0.2.0
2024-03	SA#103	SP-240252				Draft after editHelp review and presented to SA plenary for information and approval	1.0.0
2024-03	SA#103	SP-240414				Revised the Scope clause	1.1.0
2024-03	SA#103					Upgrade to change control version	18.0.0

History

Document history		
V18.0.0	May 2024	Publication