Free-Form Boolean Expression Language

The Free-Form Boolean expression language enables you to define restrictions and relationships between Variables. It is based on the Aras Markup Language (AML). The following nodes are used in the language:

expression

- eq
- variable
- named-constant
- and
- or
- not
- implication
- condition
- consequence
- exactly-one
- at-most-one
- at-least-one

Some of these nodes contain required attributes and some of them have a strict structure or required nodes.

When using an expression in AML, it should be represented as a value of a Container Node. This can be achieved in two ways: wrapped with CDATA or encoded.

CDATA Example:

```
<containerNode>
<![CDATA[<expression>
    <and>
        <eq>
            <variable id="item_id_color" />
            <named-constant id="item_id_red" />
        </eq>
        <eq>
            <variable id="item_id_wheelsize" />
            <named-constant id="item_id_17inch" />
        </eq>
    </and>
</expression>]]>
</containerNode>
```

Important

There can only be one CDATA node within a Container Node. The CDATA node can only contain one expression node.

Encoded Example:

```
<containerNode>
  <expression>
    <and>
      <eq>
        <variable id="item_id_color" />
        <named-constant id="item_id_red" />
      </eq>
      <eq>
        <variable id="item_id_wheelsize" />
        <named-constant id="item_id_17inch" />
      </eq>
    </and>
  </expression>
</containerNode>
```

The following sections describe these nodes and include examples.

Expression Nodes

Dependencies between expression nodes are listed in the following table.

Tag	Can contain
expression	implication, and, or, not, eq, exactly-one, at-most-one, at-least-one
eq	variable, named-constant
variable	id attribute

named-constant	id attribute
and	implication, and, or, not, eq, exactly-one, at-most-one, at-least-one
or	implication, and, or, not, eq, exactly-one, at-most-one, at-least-one
not	implication, and, or, not, eq, exactly-one, at-most-one, at-least-one
implication	condition, consequence
condition	implication, and, or, not, eq, exactly-one, at-most-one, at-least-one
consequence	implication, and, or, not, eq, exactly-one, at-most-one, at-least-one
exactly-one	eq
at-least-one	eq
at-most-one	eq

<expression>...</expression>

The `<expression>` node is a root node that represents the expression. Use the `<expression>` tag to define any expression in Boolean language.

```
<expression>
  <and>
    <eq>
      <variable id="item_id_color" />
      <named-constant id="item_id_red" />
    </eq>
    <eq>
      <variable id="item_id_wheelsize" />
      <named-constant id="item_id_17inch" />
    </eq>
  </and>
</expression>
```

This example represents the Boolean expression for Color = Red AND WheelSize = 17 inch.

<eq>...</eq>

The `<eq>` node defines equivalence. Equivalence can be done between Variable and NamedConstant. `<eq>` has a strict content: it must include the node pair `<variable>` and `<named-constant>` .

```
<eq>
  <variable id="item_id_color" />
  <named-constant id="item_id_red" />
</eq>
```

This example represents the Boolean expression for Color = Red.

<variable id="..."/>

The `<variable>` node defines a variable element. Use this element to define the first part of an equivalence. The first part of an equivalence should always be the variable. The “id” attribute is required; it defines the unique identifier for the instance.

```
<eq>
  <variable id="item_id_color" />
  <named-constant id="item_id_red" />
</eq>
```

This example represents the Boolean expression for Color = Red where Color is the variable.

<named-constant id="..."/>

The `<named-constant>` node defines the namedConstant element. Use this element to define the second part of equivalence. The second part of equivalence should always be namedConstant. The “id” attribute is required; it defines the unique identifier for the instance.

```
<eq>
  <variable id="item_id_color" />
  <named-constant id="item_id_red" />
</eq>
```

This example represents the Boolean expression for Color = Red where Red is the named constant.

<and>...</and>

The `<and>` node that defines the Boolean operation “and”. It can contain an unlimited number of child tags.

> **Important**
>
> It is unnecessary to include the <and></and> node explicitly within the expression node because is already implied.

```
<and>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_red" />
  </eq>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_green" />
  </eq>
</and>
```

This example represents the Boolean expression for Color = Red AND Color = Green.

<or>...</or>

The `<or>` node defines the Boolean operation “or”. It can contain an unlimited number of child tags.

```
<or>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_red" />
  </eq>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_green" />
  </eq>
</or>
```

This example represents the Boolean expression for Color = Red OR Color = Green.

The following is an example of an OR node using an inner AND node:

```
<or>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_red" />
  </eq>
  <and>
    <eq>
      <variable id="item_id_wheelsize" />
      <named-constant id="item_id_17inch" />
    </eq>
    <eq>
      <variable id="item_id_color" />
      <named-constant id="item_id_green" />
    </eq>
  </and>
</or>
```

This example represents the Boolean expression for Color = Red OR (WheelSize = 17inch AND Color = Green).

<not>...</not>

The `<not>` node defines the Boolean operation “not”. It can contain an unlimited number of child tags.

```
<not>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_red" />
  </eq>
</not>
```

This example represents the Boolean expression for NOT(Color = Red).

```
<not>
  <or>
    <eq>
      <variable id="item_id_wheelsize" />
      <named-constant id="item_id_17inch" />
    </eq>
    <eq>
      <variable id="item_id_color" />
      <named-constant id="item_id_green" />
    </eq>
  </or>
</not>
```

This example represents the Boolean expression for NOT(WheelSize = 17 inch OR Color = Green). This is the same as NOT(WheelSize = 17 inch) AND NOT(Color = Green).

Important

When an explicit tag to wrap child tags is not provided, the child tags are wrapped by the ***and*** operator. It is explained in below example.

The following example has multiple child nodes contained within the `<not>` node:

```
<not>
  <eq>
    <variable id="item_id_wheelsize" />
    <named-constant id="item_id_16inch" />
  </eq>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_red" />
  </eq>
</not>
```

The above example is equivalent to the case when child nodes are wrapped by `<and>` node as below:

```
<not>
  <and>
    <eq>
      <variable id="item_id_wheelsize" />
      <named-constant id="item_id_16inch" />
    </eq>
    <eq>
      <variable id="item_id_color" />
      <named-constant id="item_id_red" />
    </eq>
  </and>
</not>
```

This example represents the Boolean expression for NOT(WheelSize = 16 inch AND Color = Red). This is the same as NOT(WheelSize = 16 inch) OR NOT(Color = Red).

<implication>...</implication>

The `<implication>` node defines a Boolean operation “implication” for example “if Color = Red then WheelSize = 17inch”. This node should always contain the `<condition>` and `<consequence>` child nodes. The order of the child nodes is important: the `<condition>` node must always precede the `<consequence>` node. These nodes should not be left empty.

```
<implication>
  <condition>
    <eq>
      <variable id="item_id_color" />
      <named-constant id="item_id_red" />
    </eq>
  </condition>
  <consequence>
    <eq>
      <variable id="item_id_wheelsize" />
      <named-constant id="item_id_17inch" />
    </eq>
  </consequence>
</implication>
```

This example represents the Boolean expression for IF Color = Red THEN WheelSize = 17inch.

Important

When an explicit tag to wrap child tags within the `<condition>` or `<consequence>` nodes is not provided, the child tags are wrapped by ***and*** operator. It is explained in below example.

The following example shows multiple child nodes contained within the `<condition>` and `<consequence>` nodes:

```
<implication>
  <condition>
    <eq>
      <variable id="item_id_color" />
      <named-constant id="item_id_red" />
    </eq>
    <eq>
      <variable id="item_id_type" />
      <named-constant id="item_id_mountain" />
    </eq>
  </condition>
  <consequence>
    <eq>
      <variable id="item_id_wheelsize" />
      <named-constant id="item_id_16inch" />
    </eq>
    <eq>
      <variable id="item_id_framesize" />
      <named-constant id="item_id_large" />
    </eq>
  </consequence>
</implication>
```

The above example is equivalent to the case when they are wrapped by `<and>` node as shown below:

```
<implication>
  <condition>
    <and>
      <eq>
        <variable id="item_id_color" />
        <named-constant id="item_id_red" />
      </eq>
      <eq>
        <variable id="item_id_type" />
        <named-constant id="item_id_mountain" />
      </eq>
    </and>
  </condition>
  <consequence>
    <and>
      <eq>
        <variable id="item_id_wheelsize" />
        <named-constant id="item_id_16inch" />
      </eq>
      <eq>
        <variable id="item_id_framesize" />
        <named-constant id="item_id_large" />
      </eq>
    </and>
  </consequence>
</implication>
```

This example represents the Boolean expression for IF (Color = Red AND Type = Mountain) THEN WheelSize = 16inch AND FrameSize = Large.

<exactly-one/>

The `<exactly-one>` node defines an operation. The operator describes a condition that means “one and only one of the listed equivalencies can be true”. `<exactly-one>` should contain a set of `<eq>` child nodes.

```
<exactly-one>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_red" />
  </eq>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_green" />
  </eq>
</exactly-one>
```

This example represents the Boolean expression for EXACTLY-ONE(Color = Red | Color = Green).

<at-most-one/>

The `<at-most-one>` node defines an operation. The operator describes a condition that means “either none or just one of the listed equivalencies can be true”. `<at-most-one>` should contain a set of `<eq>` child nodes.

```
<at-most-one>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_red" />
  </eq>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_green" />
  </eq>
</at-most-one>
```

This example represents the Boolean expression for AT-MOST-ONE(Color = Red | Color = Green).

<at-least-one>

The `<at-least-one>` node defines an operation. The operator describes a condition that means “at least one of the listed equivalencies should be true”. `<at-least-one>` should contain a set of `<eq>` child nodes:

```
<at-least-one>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_red" />
  </eq>
  <eq>
    <variable id="item_id_color" />
    <named-constant id="item_id_green" />
  </eq>
</at-least-one>
```

This example represents the Boolean expression for AT-LEAST-ONE(Color = Red | Color = Green).