Free-Form Boolean Expression Language

The Free-Form Boolean expression language enables you to easily 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
- ge
- le
- variable
- named-constant
- 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_model" />
            <named-constant id="item_id_z5" />
          </eq>
          <eq>
            <variable id="item_id_unit" />
            <constant type="int">10</constant>
          </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>
&lt;expression&gt;
        &lt;and&gt;
          &lt;eq&gt;
            &lt;variable id="item_id_model" /&gt;
            &lt;named-constant id="item_id_z5" /&gt;
          &lt;/eq&gt;
          &lt;eq&gt;
            &lt;variable id="item_id_unit" /&gt;
            &lt;constant type="int"&gt;10&lt;constant&gt;
          &lt;/eq&gt;
        &lt;/and&gt;
&lt;/expression&gt;
</containerNode>
```

The following sections describe these nodes and include examples.

Expression Nodes

Table below lists dependencies between expression nodes.

Dependencies between expression nodes

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

not	implication, and, or, not, eq, ge, le, exactly-one, at-most-one, at-least-one
implication	condition, consequence
condition	implication, and, or, not, eq, ge, le, exactly-one, at-most-one, at-least-one
consequence	implication, and, or, not, eq, ge, le, 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 an expression in Boolean language.

```
<expression>
  <and>
  <eq>
   <variable id="item_id_model" />
   <named-constant id="item_id_z5" />
  </eq>
  <le>
    <variable id="item_id_unit" />
    <constant type="int">10</constant>
  </le>
  </and>
</expression>
```

This example represents the Boolean expression for Model = Z5 AND Unit <= 10.

<eq>…</eq>

The `<eq>` node defines equivalence. Equivalence is defined between a Variable and a Named Constantor Constant. `<eq>` has a strict content: it must include the node pair `<variable>` and `<named-constant>` or `<constant>` .

```
<eq>
  <variable id="item_id_model" />
  <named-constant id="item_id_z5" />
</eq>
```

This example represents the Boolean expression for Model = Z5.

```
<eq>
  <variable id="item_id_unit" />
  <constant type="int">10</constant>
</eq>
```

This example represents the Boolean expression for Unit = 10.

<ge>…</ge>

The `<ge>` node defines a greater than or equal to operator. This operator is defined between a Variable and a Constant. `<ge>` has a strict content: it must include the node pair `<variable>` and `<constant>` .

```
<ge>
  <variable id="item_id_unit" />
  <constant type="int">1</constant>
</ge>
```

This example represents the Boolean expression for Unit >= 10.

<le>…</le>

The `<le>` node defines a less than or equal to operator. This operator is defined between a Variable and a Constant. `<le>` has a strict content: it must include the node pair `<variable>` and `<constant>` .

```
<le>
  <variable id="item_id_unit" />
  <constant type="int">10</constant>
</le>
```

This example represents the Boolean expression for Unit <= 10.

<variable id=”…”/>

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

```
<eq>
  <variable id="item_id_model" />
  <named-constant id="item_id_z5" />
</eq>
```

This example represents the Boolean expression for Model = Z5 where Model is the variable.

<named-constant id=”…”/>

The `<named-constant>` node defines the namedConstant element. This element is used to define the second part of an equivalence. The “id” attribute is required; it defines the unique identifier for the namedConstant instance. NamedConstant is used when a variable can have a value from a list of defined values.

```
<eq>
  <variable id="item_id_model" />
  <named-constant id="item_id_z5" />
</eq>
```

This example represents the Boolean expression for Model = Z5 where Z5 is the named constant.

<constant>…</constant>

The `<constant>` node defines a constant element. This element is used to define the second part of an operation. The “type” attribute is required; it defines the Constant value type. Supported types are as follows:

- Int
- DateTime
- String

```
<le>
  <variable id="item_id_unit" />
  <constant type="int">10</constant>
</le>
```

This example represents the Boolean expression for Unit <= 10 where 10 is the constant.

<and>…</and>

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

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

```
<and>
  <eq>
    <variable id="item_id_model" />
    <named-constant id="item_id_z5" />
  </eq>
  <le>
    <variable id="item_id_unit" />
    <constant type="int">10</constant>
  </le>
</and>
```

This example represents the Boolean expression for Model = Z5 AND Unit <= 10.

<or>…</or>

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

```
<or>
  <eq>
    <variable id="item_id_moel" />
                <named-constant id="item_id_z5" />
  </eq>
  <le>
    <variable id="item_id_unit" />
    <constant type="int">10</constant>
  </le>
</or>
```

This example represents the Boolean expression for Model = Z5 OR Unit <= 10.
The following is an example of an OR node using an inner AND node:

```
<or>
  <eq>
    <variable id="item_id_model" />
    <named-constant id="item_id_z5" />
  </eq>
  <and>
    <ge>
      <variable id="item_id_unit" />
      <constant type="int">10</constant>
    </ge>
    <eq>
      <variable id="item_id_model" />
      <named-constant id="item_id_z6" />
    </eq>
  </and>
</or>
```

This example represents the Boolean expression for Model = Z5 OR (Unit >= 10 AND Model = Z6).

<not>…</not>

The `<not>` node defines the Boolean operation “not”. Only one child node can be created within this node. The child node can contain an unlimited number of nested, allowed child nodes, as shown in the second example.

```
<not>
  <eq>
    <variable id="item_id_model" />
    <named-constant id="item_id_z5" />
  </eq>
</not>
```

This example represents the Boolean expression for NOT(Model = Z5).

The following example shows that the child node contained within the <not> node can have nested child nodes:

```
<not>
  <and>
    <eq>
      <variable id="item_id_model" />
      <named-constant id="item_id_z5" />
    </eq>
    <eq>
      <variable id="item_id_unit" />
      <constant type="int">10</constant>
    </eq>
  </and>
</not>
```

This example represents the Boolean expression for NOT(Model = Z5 AND Unit = 10). It is different than NOT(Model = Z5) OR NOT(Unit = 10).

<implication>…</implication>

The `implication` node defines a Boolean operation “implication”, such as “if Model = Z6 then Unit >= 10”. This node must contain the `<condition>` and `<consequence>` child nodes. The `<condition>` node must precede the `<consequence>` node. Neither of these nodes can be empty.

```
<implication>
  <condition>
    <eq>
      <variable id="item_id_model" />
      <named-constant id="item_id_z6" />
    </eq>
  </condition>
  <consequence>
    <ge>
      <variable id="item_id_unit" />
      <constant type="int">10</constant>
    </ge>
  </consequence>
</implication>
```

This example represents the Boolean expression for IF Model = Z6 THEN Unit >= 10.

You can have an unlimited number of child nodes in condition and consequence tags as shown in the example.

The following example shows that `condition` and `consequence` nodes can be used with inner `and` and `or` nodes:

```
<implication>
  <condition>
    <or>
      <eq>
        <variable id="item_id_model" />
        <named-constant id="item_id_z5" />
      </eq>
      <eq>
        <variable id="item_id_model" />
        <named-constant id="item_id_z6" />
      </eq>
   </or>
  </condition>
  <consequence>
    <and>
      <ge>
        <variable id="item_id_unit" />
        <constant type="int">10</constant>
      </ge>
      <le>
        <variable id="item_id_unit" />
        <constant type="int">20</constant>
      </le>
    </and>
  </consequence>
</implication>
```

This example represents the Boolean expression for IF (Model = Z5 OR Model = Z6) THEN (Unit >= 10 AND Unit <= 20).

<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>` mustcontaina set of `<eq>` child nodes.

```
<exactly-one>
  <eq>
    <variable id="item_id_model" />
    <named-constant id="item_id_z5" />
  </eq>
  <eq>
    <variable id="item_id_model" />
    <named-constant id="item_id_z6" />
  </eq>
</exactly-one>
```

This example represents the Boolean expression for EXACTLY-ONE(Model = Z5 | Model = Z6).

Important

Each equivalence in EXACTLY-ONE must use the same variable.

<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>` must contain a set of `<eq>` child nodes.

Each equivalence in AT-MOST-ONE must use the same variable.

```
<at-most-one>
  <eq>
    <variable id="item_id_model" />
    <named-constant id="item_id_z5" />
  </eq>
  <eq>
    <variable id="item_id_model" />
    <named-constant id="item_id_z6" />
  </eq>
</at-most-one>
```

This example represents the Boolean expression for AT-MOST-ONE(Model = Z5 | Model = Z6).

Important

Each equivalence in AT-MOST-ONE must use same variable.

<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”. must contain a set of child nodes:

Each equivalence in AT-LEAST-ONE must use the same variable.

```
<at-least-one>
  <eq>
    <variable id="item_id_model" />
    <named-constant id="item_id_z5" />
  </eq>
  <eq>
    <variable id="item_id_model" />
    <named-constant id="item_id_z6" />
  </eq>
</at-least-one>
```

This example represents the Boolean expression for AT-LEAST-ONE(Model = Z5 | Model = Z6).

Important

Each equivalence in AT-LEAST-ONE must use the same variable.

Effectivity Criteria Expression Format

Effectivity Criteria supports only direct child nodes under node. It can have one or more nodes, but they cannot be grouped using any grouping operators. Otherwise, Effectivity Services core will throw an exception. Each equivalence node can contain only a Variable and a NamedConstant or Constant node.

```
<expression>
  <eq>
    <variable id="item_id_model" />
    <named-constant id="item_id_z5" />
  </eq>
  <eq>
    <variable id="item_id_unit" />
    <constant type="int">10</constant>
  </eq>
</expression>
```