Aras Innovator 11

Configurator Services

Programmer's Guide

Document #: 11.12.02018122901

Last Modified: *9/21/2020*

Copyright Information

Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810

Phone: 978-691-8900
Fax: 978-794-9826

E-mail: support@aras.com

Website: https://www.aras.com/

Table of Contents

Send Us Your Comments

Aras Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for future revisions.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where and what level of detail?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, indicate the document title, and the chapter, section, and page number (if available).

You can send comments to us in the following ways:

Email:
Support@aras.com
Subject: Aras Innovator Documentation

Or,

Postal service:
Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Attention: Aras Innovator Documentation

Or,

FAX:
978-794-9826
Attn: Aras Innovator Documentation

If you would like a reply, provide your name, email address, address, and telephone number.

If you have usage issues with the software, visit https://www.aras.com/support

Document Conventions

The following table highlights the document conventions used in the document:

Table 1: Document Conventions

Convention	Description
Bold	Emphasizes the names of menu items, dialog boxes, dialog box elements, and commands. Example: Click **OK**.
`Code`	Code examples appear in `courier` font. It may represent text you type or data you read.
`Yellow highlight`	Code highlighted in yellow draws attention to the code that is being indicated in the content.
`Yellow highlight with red text`	Red text highlighted in yellow indicates the code parameter that needs to be changed or replaced.
Italics	Reference to other documents.
Note:	Notes contain additional useful information.
Warning	Warnings contain important information. Pay special attention to information highlighted this way.

1 Introduction

Rapid technological advances and high customer expectations have been driving companies to build highly configurable products with an increasing number of product features and options in all domains of the product.

A product variant is a specific configuration of product for certain option choices, such as size Medium and color Red. The complexity of product variants grows exponentially with the number of features available in the product, and the number of options within each feature. The complexity is multiplied by the time dimension, where new features and options are introduced, or existing features and options are obsoleted. Constraints, driven by Marketing, Engineering, etc. organizations, restrict the set of allowed or compatible option combinations.

Configurator Services, built into the Aras PLM Platform, is the enabler to develop variant management applications to manage product complexity. The configurator uses a rule-based configuration paradigm, and has a built-in Boolean Satisfiability (SAT) Solver.

Using Configurator Services, customers have the ability to define any data model and business logic to meet business processes. For instance, companies may be managing their products by platforms, product lines, product models, systems/subsystems, or other product groupings. Product features, options and constraints/rules can be defined at all relevant levels for these structures. Configurator Services provides the means to implement server method to translate business data into input data for configurator APIs.

Using Configurator Services, you can:

- ✓ Validate variability definitions
- ✓ Solve variability problems to find a list of valid options
- ✓ Get reasons for invalid options

2 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
- 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>
```

Note: 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_color" /&gt;
            &lt;named-constant id="item_id_red" /&gt;
          &lt;/eq&gt;
          &lt;eq&gt;
            &lt;variable id="item_id_wheelsize" /&gt;
            &lt;named-constant id="item_id_17inch" /&gt;
          &lt;/eq&gt;
        &lt;/and&gt;
    &lt;/expression&gt;
</containerNode>
```

The following sections describe these nodes and include examples.

2.1 Expression Nodes

Dependencies between expression nodes are listed in table below.

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

Tag	Can contain
exactly-one	eq
at-most-one	eq
at-least-one	eq

2.1.1 <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.

2.1.2 <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.

2.1.3 <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.

2.1.4 <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.

2.1.5 <and>…</and>

The `<and>` node that defines the Boolean operation "and". You can create an unlimited number of child tags.

Note: 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.

2.1.6 <or>…</or>

The `<or>` node defines the Boolean operation "or". You can create 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).

2.1.7 <not>…</not>

The `<not>` node defines the Boolean operation "not". You can create just one child node. This child node can contain any number of nested child nodes, as shown in the second example.

```
<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).

The following example shows that the child node contained within the <not> node can contain an unlimited number of nested child nodes:

```
<not>
    <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>
</not>
```

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

2.1.8 <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. You should not leave either of these nodes empty. You can have an unlimited number of condition and consequence child nodes as shown in the second example.

```
<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.

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_color" />
                <named-constant id="item_id_red" />
            </eq>
            <eq>
                <variable id="item_id_color" />
                <named-constant id="item_id_green" />
            </eq>
        </or>
    </condition>
    <consequence>
        <and>
            <eq>
                <variable id="item_id_wheelsize" />
                <named-constant id="item_id_17inch" />
            </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 OR Color = Green THEN WheelSize = 17inch AND FrameSize = Large.

2.1.9 <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).

2.1.10 <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).

2.1.11 <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).

3 Brief Description of the API

The Configurator Services API consists of several methods. This section describes the purpose of these methods.

Configurator Services works with any custom business model. The Scope Builder method takes the customer data and translates it into a Scope object that is used by the Configurator Services API. The Scope object is used by the Configurator Services API to solve the appropriate task.

The following API methods use a scope object:

- **cfs_getScopeStucture**. builds the AML representation of the Scope object. It builds the Scope object with the help of scope_builder and serializes the Scope object to AML. It can be used:
    - to display a list of available choices
    - to get actual data inside implementation of new server methods
    - for debugging purposes
- **cfg_getValidCombinations**. This method is designed to find a list of valid combinations, where a combination is a list of Variables with assigned values. It can be used to:
    - find all valid combinations
    - find at least one valid combination
    - validate the current scope
    - validate values selected for Variables
    - validate an expression
- **cfg_validateScope**. This method is designed to validate the Scope. The method has two validations:
    - if Scope has at least one valid combination
    - if each value can be assigned to a Variable for at least one valid combination

  The method can be used to:
    - find out if the specified Scope is solvable
    - determine if unreachable values exist
- **cfg_getIntersectingExpressions**. This method is designed to find the intersections of expressions. Expressions are intersecting if the Scope has at least one valid combination with specified expressions applied.
- **cfg_getConflicts**. This method is designed to find the reason why the current Scope is unsolvable. The method can be used to:
    - get data that describes reasons
    - debug why you have no solutions

4 Scope Object Model

Configurator Services has an internal Scope object model that is customizable. Configurator Services processing logic is built around this object model.

This section describes the following:

- Scope Object Model
- Configurator API Method Work Flow
- Building the Scope Object Model

4.1 Scope Object Model Design

The Scope object contains Variables with a list of values that can be assigned to the variable and rules that define relationships between assigned variables.

The following diagram illustrates the structure of the Scope Object Model:

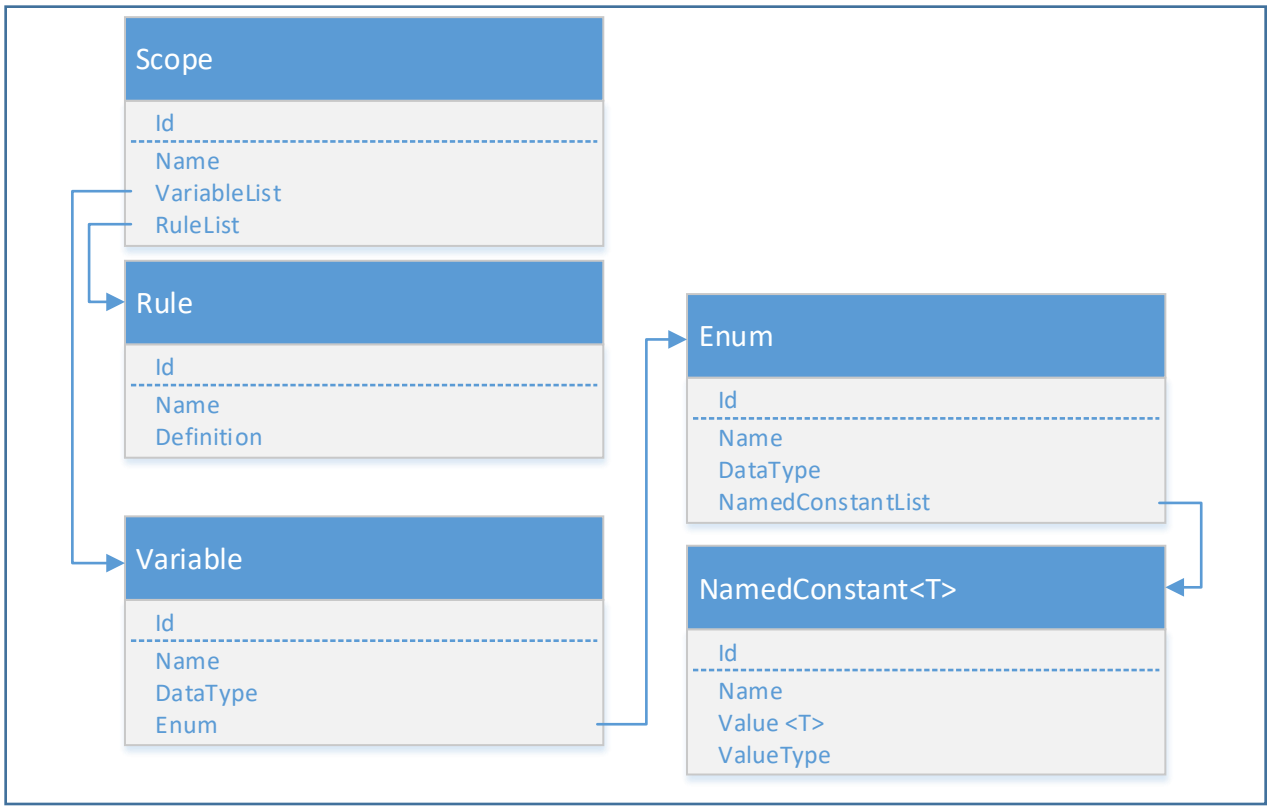


Figure 1.

- Scope is a class that contains two lists: a list of Variables and a list of Rules.
- Variable class is intended to store the variable definition.

- Named constant is intended to store the value definition. It contains the property name, value, and valuetype.
- Enum is the container for the list of Named constants.
- Rule is a container for an expression. The Rule has both a name and definition where the definition type is ExpressionBase.
- ExpressionBase is an object representation of a Boolean expression.

4.2 Configurator API Method Workflow

The following workflow diagram illustrates the process of collecting data from its source and using this data to perform an API Action.

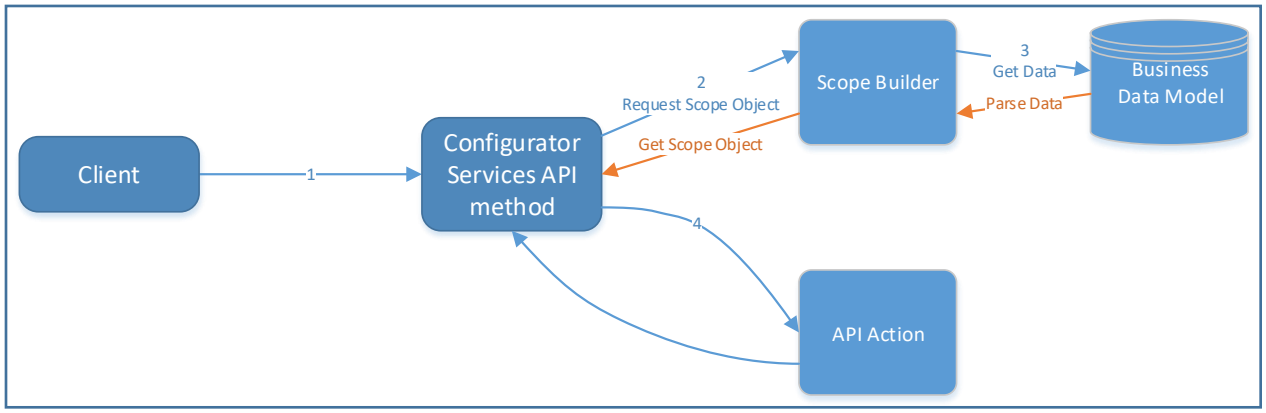


Figure 2.

1. The Client calls the Configurator Services API method.
2. The Configurator Services API method requests the scope builder method to build the Scope object.
3. The scope builder method collects and parses data from the business data source.
4. The Configurator Services API method performs an API action in the context of the built Scope object.

4.3 Customizing Scope Builder

Scope Builder is a server method that uses a predefined method template. The template enables customers to implement a Scope builder method. All of the Configurator Services API methods that perform actions in the context of Scope should be parameterized with the Scope object. The following is an example of the method template:

```
<Item type="Method" action="%action name%" %specific API attributes%>
    <targetScope>
        <Item type="Method" action="%builder method name%" %builder method attributes%>
            %builder method properties%
        </Item>
    </targetScope>
```

```
   %specific API properties%
</Item>
```

4.3.1 Implementing the Scope Builder Method

The Scope Builder method uses the CSharp:Aras.Server.Core.Configurator template to define a class that inherits from the base abstract class. The following is the template for the Scope builder method:

```
namespace $(pkgname)
{
    public class $(clsname): ScopeBuilderBase
    {
        $(MethodCode)
    }
}
```

Base abstract class:

```
public abstract class ScopeBuilderBase
{
        public Item ScopeItem { get; set; }
        public IServerConnection IomConnection { get; private set; }
        public void Init(Item scopeItem, IServerConnection iomConnection) {}
        public abstract Scope BuildScope();
        public abstract string[] GetGuidsItemDependsOn();
        public abstract List<string> GetItemTypeNamesItemDependsOn();
        public abstract ArrayList GetCustomKey();
}
```

The Scope Builder method is designed to provide a way for constructing a Scope object. The Scope Builder method also provides the built-in possibility to cache the Scope object.

Note: Anyone who implements a scope builder method will be forced to override all abstract methods.

4.3.2 Input Item Format

The targetScope Property enables you to specify the Scope builder method for Configurator Services API methods using AML syntax.

The syntax is as follows:

```
<Item type="Method" action="%action name%" %specific API attributes%>
   <targetScope>
      <Item type="Method" action="%builder method name%" %builder method attributes%>
         %builder method properties%
      </Item>
   </targetScope>
   %specific API properties%
</Item>
```

Note: Samples will be provided for each method in "5 AML API Methods".

Where:

%*action name*%	Name of the action associated with the Configurator API method.
%*specific API attribute*%	List of API attributes specifically for a particular method. For example, "fetch" is an API attribute for the GetValidCombination method.
%*builder method name*%	Name of the server method that builds the Scope object model based on the business logic being used.
%*builder method attributes*%	Attributes that are needed in order for the builder method to process a business Item.
%*builder method properties*%	Properties that are needed for the builder method to process the business item.
%*specific API properties*%	List of the API properties specific to a particular method. For example, the "condition" property is specific to the GetValidCombinations method.

%*specific API attributes*% and **%*specific API properties*%** define the method signature that is specified in the action attribute.

%*builder method attributes*% and **%*builder method properties*%** should contain all the information that is necessary for building the Scope object.

4.3.3 Scope Resolver

Scope Resolver is an internal module that is used as part of the Scope Builder process. It is used to parse request AML, call for the Scope builder method, and cache the Scope builder method result. The following description is for information only, to make internal processing clearer:

```
public class ScopeResolverModule
    {
        public Scope BuildScope(Item item)
        {
            ScopeBuilderBase builder = GetBuilder(item);
            builder.Init(item, this.serverConnection);
            return this.cache.CacheDriverNotNull(
                CachableScopeContainer.GetKey(builder.GetCustomKey()), list =>
                CachableScopeContainer.GetInstance(builder, this.cacheItemType)).Scope;
        }
        private ScopeBuilderBase GetBuilder(Item scopeItem) { }
    }
```

4.3.4 Caching

The Scope builder approach provides the built-in ability to cache a built Scope object. You need to override the CustomKey function in order to store and retrieve the scope object from the cache. This function should return the same ArrayList for each request to the same Scope object. To store a different cache for different Scope objects, the GetCustomKey function should return the different contents of the ArrayList:

```
public override ArrayList GetCustomKey()
    {
        return new ArrayList {
                ScopeItem.getID(),
            };
    }
```

The Scope object can be invalidated automatically. You need to create the following collections to make this possible:

- List of ItemType Names. This list contains the name of each Item Type that was used to build the Scope.

```
public override List<string> GetItemTypeNamesItemDependsOn()
{
    return new List<string> {
            "ItemType1",
            "ItemType2",
            "ItemType3",
            "ItemType4"
    };
}
```

  Each Item Type in this list should be a polysource item for ScopeCacheDependency.

  - List of Item IDs. This list contains the ID of each Item that was used to build the Scope.

```
public override string[] GetGuidsItemDependsOn()
{
    return new string[] {
            "item_id_color",
            "item_id_red",
            "item_id_wheelsize",
            "item_id_17inch"
    };
}
```

5 AML API Methods

The following sections describe how to use AML within the Configurator Services API. This section describes the following topics in detail:

- AML Request Syntax
- AML Response Formats
- AML Request/Response examples

5.1 GetScopeStructure

The GetScopeStructure method is designed to show the Scope Structure in AML format.

The output allows customizations to include extended information about Items.

5.1.1 AML Request Syntax

```
<Item type="Method" action="cfg_GetScopeStructure">
    <targetScope>
        <Item type="Method" action="%builder method name%" %builder method attributes% >
            %builder method properties%
        </Item>
    </targetScope>
    <output_buider_method>%output_buider_method%</output_buider_method>
</Item>
```

targetScope ***%builder method name%*** ***%builder method attributes%*** ***%builder method properties%***	Refer to "4.3.2 Input Item Format" targetScope is required.
%output_builder_method%	Optional parameter. This is the Name of the server method. This server method overrides GetScopeStructureOutputBase using the Aras.Server.Core.Configurator.ScopeStructureOutput template. The server method is used to extend the response using custom data. The AML targetScope item is accessible in the output_builder_method.

5.1.2 AML Response Format

The following response represents the Scope structure in AML format:

```
<Item type="Scope" id="ScopeId">
    <name>ScopeName</name>
    <Relationships>
```

```
        <Item type="Variable" id="VariableId_1">
            <id>VariableId_1</id>
            <name>VariableName_1</name>
            <Relationships>
                <Item type="NamedConstant" id="NamedConstantId_1">
                    <id>NamedConstantId_1</id>
                    <name>NamedConstantName_1</name>
                </Item>
                <Item type="NamedConstant" id="NamedConstantId_2">
                    <id>NamedConstantId_2</id>
                    <name>NamedConstantName_2</name>
                </Item>
            </Relationships>
        </Item>
        ...
        <Item type="Rule" id="RuleId_1">
            <id>RuleId_1</id>
            <name>RuleName_1</name>
            <definition>expression</definition>
        </Item>
        ...
    </Relationships>
</Item>
```

5.1.3 Examples

```
<Item type="Scope" id="item_id_componentA">
    <name>component A</name>
    <Relationships>
        <Item type="Variable" id="item_id_color">
            <id>item_id_color</id>
            <name>Color</name>
            <Relationships>
                <Item type="NamedConstant" id="item_id_red">
                    <id>item_id_red</id>
                    <name>Red</name>
                </Item>
                <Item type="NamedConstant" id="item_id_green">
                    <id>item_id_green</id>
                    <name>Green</name>
                </Item>
            </Relationships>
        </Item>
        <Item type="Variable" id="item_id_wheelsize">
            <id>item_id_wheelsize</id>
            <name>Wheel Size</name>
            <Relationships>
                <Item type="NamedConstant" id="item_id_15inch">
                    <id>item_id_15inch</id>
                    <name>15 inch</name>
                </Item>
                <Item type="NamedConstant" id="item_id_16inch">
                    <id>item_id_16inch</id>
                    <name>16 inch</name>
                </Item>
            </Relationships>
```

```
        </Item>
        <Item type="Rule" id="item_id_rule1">
            <id>item_id_rule1</id>
            <name>Rule Name 1</name>
            <definition>expression</definition>
        </Item>
    </Relationships>
</Item>
```

5.2 GetValidCombinations

This method is designed to find a list of valid combinations.

Parameters

Required	Optional
targetScope	fetch, offset, select, responseFormat (default is JSON), condition

5.2.1 AML Request Syntax

The AML request syntax is as follows:

```
<Item>
    type="Method"
    action="cfg_GetValidCombinations"
    select="%variableIds%"
    offset="%offset_integer%"
    fetch="%fetch_integer%"
    responseFormat="%XML|JSON%">
    <targetScope>
        <Item type="Method" action="%builder method name%" %builder method attributes%>
        %builder method properties%
        </Item>
    </targetScope>
    <condition>
        <![CDATA[
        <expression>...</expression>
        ]]>
    </condition>
</Item>
```

Where:

%variableIds%	Comma-separated variable Id list. Use the "select" attribute to request specified variables only. By default, the "select" attribute is empty. The GetValidCombinations response contains a full list of the variables that exist in the Scope.
%offset_integer%	Defines the number of combinations that can be skipped.

	Default value is 0.
%*fetch_integer*%	Defines the number of combinations that are returned, starting from the offset position. The Default value is empty, which returns all valid combinations. It is a best practice to use fetch='1' for requests that do not need a full list of combinations.
%XML|JSON%	Specifies whether the response format is XML or JSON. The default is JSON.
<condition>…</condition>	Node that contains the expression to add to the Scope before finding combinations. The Default value is empty.
targetScope **%*builder method name*%** **%*builder method attributes*%** **%*builder method properties*%**	Refer to "4.3.2 Input Item Format" targetScope is required.

5.2.2 AML Response Format

5.2.2.1 Overview

This section describes the response format in terms of the object model.

The Response object has two properties: combinations-meta and combinations.

- Combinations-meta has two properties:
  - Variables – an array of variable objects. The Variable has an ID property.
  - Values – is an array of value objects. The Value has an ID property.
- Combinations is an array of combination objects. The Combination has a value (integer) property. This value is used as an index for the Values array.

5.2.2.2 Building list of VariableId=ValueId pairs

Each element in the combination array can be translated to VariableId=ValueId. The element Index in a combination array is an index of the appropriate elements in the Variables array. The Value of a combination element is an index of the appropriate elements in a Values array.

```
for (int index = 0; index < combination.Length; index++)
{
    string VariableId = Variables[index];
    string ValueId = Values[combination[index]];
}
```

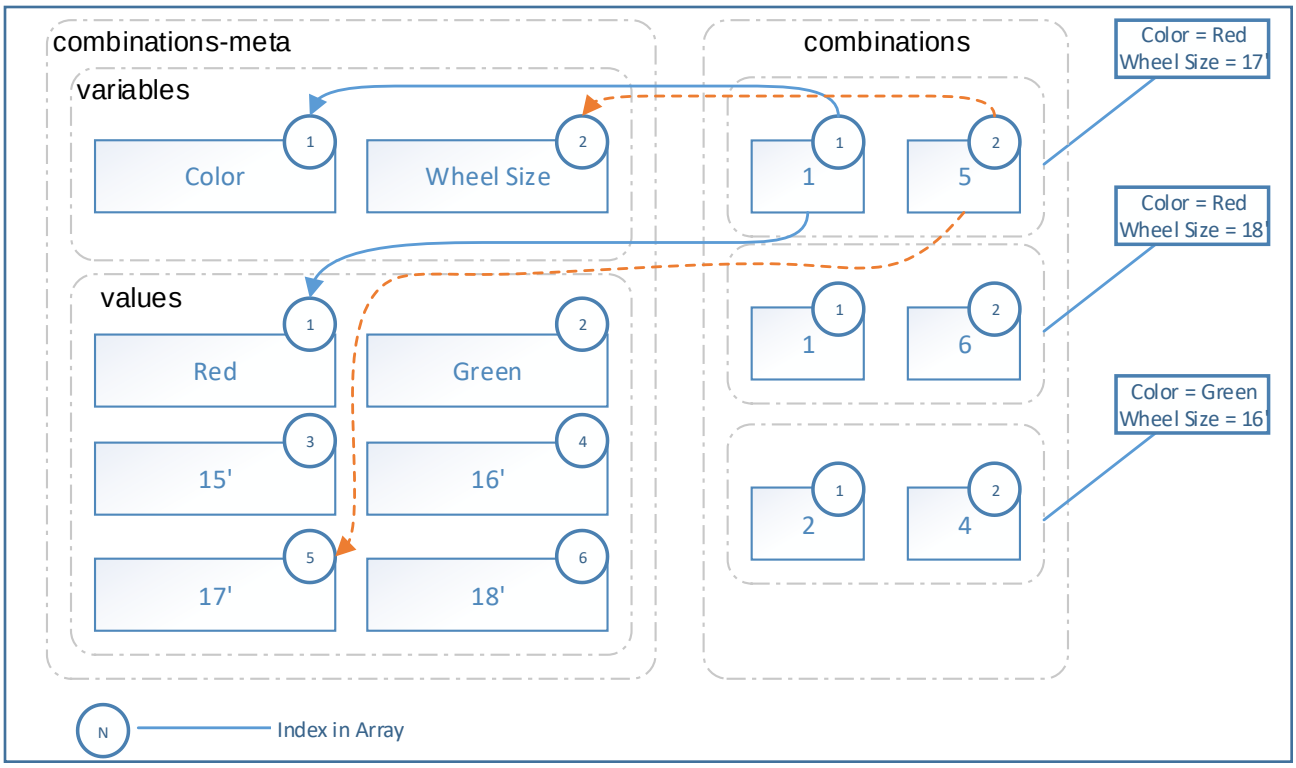


Figure 3.

5.2.2.3 XML Format

The following response in XML Format is a response object from section "5.2.2.1 Overview" serialized to XML and wrapped to the AML Result:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            <combinations-meta>
                <variables>
                    <variable id="variable0Id" order="0" />
                    <variable id="variable1Id" order="1" />
                    ...
                    <variable id="variableNId" order="N" />
                </variables>
                <values>
                    <value type="valueType" order="0">value0Id</value>
                    <value type="valueType" order="1">value1Id</value>
                    ...
                    <value type="valueType" order="M">valueMId</value>
                </values>
            </combinations-meta>
            <combinations>
                <combination>
                    <value>valueIndex1</value>
                    <value>valueIndex2</value>
                    ...
                    <value>valueIndexN</value>
```

```
                </combination>
                <combination>
                    <value>valueIndex1</value>
                    <value>valueIndex2</value>
                    ...
                    <value>valueIndexN</value>
                </combination>
                ...
                <combination>
                    <value>valueIndex1</value>
                    <value>valueIndex2</value>
                    ...
                    <value>valueIndexN</value>
                </combination>
            </combinations>
        </Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- The <combinations-meta> node contains the <variables> and <values> child nodes.
- The <variables> node consists of child <variable> nodes which are a list of variables contained in the current Scope. Each <variable> node contains the '@id' attribute which defines the variable ID, and the '@order' attribute which defines the order of the current variable within the <variables> node. Variable order starts at 0.
- The <values> node consists of a series of <value> child nodes which list all of the values contained in the current Scope. Each <value> node has a '@type' attribute and an '@order' attribute. The '@type' attribute defines the value type (for example, 'Named Constant', 'Constant'). The '@order' attribute specifies the order of the current value within the <values> node. The Value order starts at 0. The Inner text of the <value> node is the ID of the Named Constant in the current Scope.
- The '<combinations>…</combinations>' node is a list of found combinations.
- The '<combination>…</combination>' node represents one combination.
- To determine a single combination, you should use <combination> and run through the <values>. For each <value> build a "Variable = Value" pair where the Variable and the Value can be found in '<Combinations-meta></combinations-meta>.'

Each <value>…</value> node within the '<combinations-meta><values></values></combinations-meta>' must be of type Named Constant.

If there are no valid combinations, the result node will be empty:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result />
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.2.2.4 JSON Format

The response in JSON Format is a response object from section "5.2.2.1 Overview" serialized to JSON and wrapped to AML Result.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
       {
           "combinations-meta": {
               "variables": {
                   "variable0Id": 0,
                   "variable1Id": 1,
                   ...
                   "variableNId": N
               },
               "values": [{
                     "type": "valueType",
                     "id": "value0Id"
                   },
                   {
                     "type": "valueType",
                     "id": "value1Id"
                   },
                   ...
                   {
                     "type": "valueType",
                     "id": "valueMId"
                   }]
           },
           "combinations": [
               [valueIndex1,valueIndex2,...,valueIndexN],
               [valueIndex1,valueIndex2,...,valueIndexN],
               ...
               [valueIndex1,valueIndex2,...,valueIndexN]]
       }
        </Result>
    </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

- The 'combinations-meta' object has the 'variables' and 'values' objects.
- The 'variables' object consists of name/value pairs, where name is the ID of the Variable in the current Scope and value defines the order of the current variable within the 'variables' object.
- The 'values' object is an array of objects that represent the Named Constant list for the current Scope.
- The 'combinations' object represents an array of valid combinations. Each element in the array represents a single combination
- Each object within the '<values>' array is a Named Constant.

If there are no valid combinations, the result node will contain an empty JSON object:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
```

```
        <Result>{}</Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.2.3 Examples

- ***Get all valid combinations with no limits***

The request is:

```
<Item type="Method" action="cfg_GetValidCombinations" responseFormat="XML">
    <targetScope>
        <Item type="Method" action="builder_method_name" id="business_item_id"/>
    </targetScope>
</Item>
```

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <combinations-meta>
            <variables>
               <variable id="item_id_color" order="0" />
               <variable id="item_id_wheelsize" order="1" />
            </variables>
            <values>
               <value type="NamedConstant" order="0">item_id_red</value>
               <value type="NamedConstant" order="1">item_id_green</value>
               <value type="NamedConstant" order="2">item_id_15inch</value>
               <value type="NamedConstant" order="3">item_id_16inch</value>
               <value type="NamedConstant" order="4">item_id_17inch</value>
               <value type="NamedConstant" order="5">item_id_18inch</value>
            </values>
         </combinations-meta>
         <combinations>
            <combination>
               <value>0</value>
               <value>2</value>
            </combination>
            <combination>
               <value>0</value>
               <value>3</value>
            </combination>
            <combination>
               <value>1</value>
               <value>3</value>
            </combination>
            <combination>
               <value>1</value>
               <value>5</value>
            </combination>
         </combinations>
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (@responseFormat="JSON" within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
       {
          "combinations-meta": {
             "variables": {
                "item_id_color": 0,
                "item_id_wheelsize": 1
             },
             "values": [{
                "type": "NamedConstant",
                "id": "item_id_red"
             },
             {
                "type": "NamedConstant",
                "id": "item_id_green"
             },
             {
                "type": "NamedConstant",
                "id": "item_id_15inch"
             },
             {
                "type": "NamedConstant",
                "id": "item_id_16inch"
             },
             {
                "type": "NamedConstant",
                "id": "item_id_17inch"
             },
             {
                "type": "NamedConstant",
                "id": "item_id_18inch"
             }]
          },
          "combinations": [[0,2],
          [0,3],
          [1,3],
          [1,5]]
       }
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- ***Using the @fetch attribute to get the first valid combination***

The request is:

```
<Item type="Method" action="cfg_GetValidCombinations" fetch="1" responseFormat="XML">
    <targetScope>
        <Item type="Method" action="builder_method_name" id="business_item_id"/>
    </targetScope>
</Item>
```

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <combinations-meta>
            <variables>
               <variable id="item_id_color" order="0" />
               <variable id="item_id_wheelsize" order="1" />
            </variables>
            <values>
               <value type="NamedConstant" order="0">item_id_red</value>
               <value type="NamedConstant" order="1">item_id_15inch</value>

            </values>
         </combinations-meta>
         <combinations>
            <combination>
               <value>0</value>
               <value>1</value>
            </combination>
         </combinations>
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (if @responseFormat="JSON" within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
       {
          "combinations-meta": {
             "variables": {
                "item_id_color": 0,
                "item_id_wheelsize": 1
             },
             "values": [{
                "type": "NamedConstant",
                "id": "item_id_red"
             },
             {
                "type": "NamedConstant",
                "id": "item_id_15inch"
             }]
          },
          "combinations": [[0,1]]
       }
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- ***Using the @select attribute to get valid combinations for a single variable***

The request is:

```
<Item type="Method" action="cfg_GetValidCombinations" select="item_id_color"
responseFormat="XML">
    <targetScope>
        <Item type="Method" action="builder_method_name" id="business_item_id"/>
    </targetScope>
</Item>
```

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <combinations-meta>
            <variables>
               <variable id="item_id_color" order="0" />
            </variables>
            <values>
               <value type="NamedConstant" order="0">item_id_red</value>
               <value type="NamedConstant" order="1">item_id_green</value>
            </values>
         </combinations-meta>
         <combinations>
            <combination>
               <value>0</value>
            </combination>
            <combination>
               <value>1</value>
            </combination>
         </combinations>
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (@responseFormat="JSON" within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
        {
           "combinations-meta": {
              "variables": {
                 "item_id_color": 0
              },
              "values": [{
                 "type": "NamedConstant",
                 "id": "item_id_red"
              },
              {
                 "type": "NamedConstant",
                 "id": "item_id_green"
              }]
           },
           "combinations": [[0],[1]]
        }
```

```
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.3 CfgValidateScope

The CfgValidateScope method checks to see if the built Scope has at least one valid combination. It also checks to see if each value can be assigned to a Variable for at least one valid combination.

5.3.1 AML Request Syntax

```
<Item type="Method" action="cfg_ValidateScope">
    <targetScope>
        <Item type="Method" action="%builder method name%" %builder method attributes%>
            %builder method properties%
        </Item>
    </targetScope>
</Item>
```

targetScope - ***%builder method name%*** - ***%builder method attributes%*** - ***%builder method properties%***	Refer to "4.3.2 Input Item Format" targetScope is required.

5.3.2 AML Response Format

The Response contains the Validation status and a description of the error:

1. Validation passed:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            <Validation type="Validation" result="true"/>
        </Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1. Validation failed – Scope has no valid combinations:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            <Validation type="Validation" result="false">
                <Error>
                    <Name>Validate Scope</Name>
                    <Message>The problem does not have an optimal solution!</Message>
                </Error>
```

```
          </Validation>
        </Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2. Validation failed – some values can't be assigned:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <Result>
   <Validation type="Validation" result="false">
    <Error>
     <Name>Scope Named Constants Availability Validate</Name>
     <Message>2 values are not available</Message>
     <details>
      <EQ>
       <variable name="Variable1" id="item_id_color" />
       <named-constant name="Value1" id="item_id_red" />
      </EQ>
      <EQ>
       <variable name="Variable1" id="item_id_wheelsize" />
       <named-constant name="Value2" id="item_id_16inch" />
      </EQ>
     </details>
    </Error>
   </Validation>
  </Result>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.4 CfgGetIntersectingExpressions

The CfgGetIntersectingExpressions method enables you to calculate the intersections between two or more expressions.

- The least complicated case is to check to see if the specified expressions are intersecting.
- The moderately complicated case is to find a pair of expressions from a list that are intersecting.
- The most complicated case is to find a combination of expressions from a list that are intersecting.

The first thing to do to resolve these tasks is to define a full list of expression combinations that need to be checked. After defining the expression combinations, validate each expression combination.

Note: Two or more expressions are intersecting if at least one valid combination exists that satisfies each expression.

5.4.1 AML Request Syntax

The AML request syntax is as follows:

```
<Item type="Method" action="cfg_GetIntersectingExpressions" responseFormat="%XML|JSON%">
   <targetScope>
      <Item type="Method" action="%builder method name%" %builder method attributes%>
         %builder method properties%
      </Item>
   </targetScope>
   <cartesian-product>
      <set>
            <expression id="e1"><![CDATA[<expression>...</expression>]]></expression>
            <expression id="e2"><![CDATA[<expression>...</expression>]]></expression>
            ...
            <expression id="eN"><![CDATA[<expression>...</expression>]]></expression>
      </set>
      <set>
            <expression id="eE1"><![CDATA[<expression>...</expression>]]></expression>
            <expression id="eE2"><![CDATA[<expression>...</expression>]]></expression>
            ...
            <expression id="eEY"><![CDATA[<expression>...</expression>]]></expression>
      </set>
       ...
      <set>
            <expression id="eM1"><![CDATA[<expression>...</expression>]]></expression>
            <expression id="eM2"><![CDATA[<expression>...</expression>]]></expression>
            ...
            <expression id="eMZ"><![CDATA[<expression>...</expression>]]></expression>
      </set>
   </cartesian-product>
</Item>
```

You must include at least two sets within the 'cartesian-product' node

%XML|JSON%	Specifies whether the response format is XML or JSON. The default is JSON.
targetScope **%*builder method name*%** **%*builder method attributes*%** **%*builder method properties*%**	Refer to "4.3.2 Input Item Format" targetScope is required.
<cartesian-product>…</cartesian-product>	The node that contains sets of expressions that need to be verified for intersection within the provided Scope.

<set>..</set>	The container for expressions that is used to verify intersections with expressions contained within other <set>…</set> nodes. Since expressions within a single <set>…</set> node are not verified, you must provide at least two <set>…</set> nodes.

5.4.1.1 Algorithm overview

1. Take one expression from each <set>.
2. Concatenate the selected expressions in one expression using <AND>.
3. Validate the concatenated expression for the current Scope.
4. Repeat steps 1-3 for new expressions selection.

The Selected expressions are considered to be intersecting if there is at least one valid combination for Scope.

The following examples demonstrate how the algorithm prepares sets of expressions:

- In the following example, there are 2 sets and each set contains one element:

  A (a1), B (b1)

  The algorithm creates the following set:

  (a1, b1), (b1, a1)

  There are 2 possible sets. They contain the same elements, but in a different order. The final result looks like this:

  (a1, b1)

- In the following example, there are 5 sets and each of them contains one element:

  A (a1), B (b1), C (c1), D (d1), E (e1)

  The algorithm prepares the following sets:

  (a1, b1), (a1, c1), (a1, d1), (a1, e1)

  (b1, a1), (b1, c1), (b1, d1), (b1, e1)

  (c1, a1), (c1, b1), (c1, d1), (c1, e1)

  (d1, a1), (d1, b1), (d1, c1), (c1, e1)

  (e1, a1), (e1, b1), (e1, c1), (e1, d1)

  There are 20 possible sequences. Some of them contain the same elements, but they are ordered differently. The final result looks like this:

(a1, b1), (a1, c1), (a1, d1), (a1, e1)

(b1, c1), (b1, d1), (b1, e1)

(c1, d1), (c1, e1)

(d1, e1)

- In the following example, there are 3 sets and each of them contains two elements:

  A (a1, a2), B (b1, b2), C (c1, c2)

  The algorithm prepares the following sets:

  (a1, b1, c1), (a1, b1, c2), (a1, b2, c1), (a1, b2, c2)

  (a2, b1, c1), (a2, b1, c2), (a2, b2, c1), (a2, b2, c2)

  (b1, a1, c1), (b1, a1, c2), (b1, a2, c1), (b1, a2, c2)

  (b2, a1, c1), (b2, a1, c2), (b2, a2, c1), (b2, a2, c2)

  (c1, a1, b1), (c1, a1, b2), (c1, a2, b1), (c1, a2, b2)

  (c2, a1, b1), (c2, a1, b2), (c2, a2, b1), (c2, a2, b2)

  There are 24 possible sequences and some of them contain the same elements, but they are ordered differently. The final result looks like this:

  (a1, b1, c1), (a1, b1, c2), (a1, b2, c1), (a1, b2, c2)

  (a2, b1, c1), (a2, b1, c2), (a2, b2, c1), (a2, b2, c2)

5.4.1.2 Cartesian Square AML Request Syntax

```
<Item type="Method" action="cfg_GetIntersectingExpressions" responseFormat="%XML|JSON%">
   <targetScope>
      <Item type="Method" action="%builder method name%" %builder method attributes%>
         %builder method properties%
      </Item>
   </targetScope>
   <cartesian-square>
      <expression id="e1"><![CDATA[<expression>...</expression>]]></expression>
      <expression id="e2"><![CDATA[<expression>...</expression>]]></expression>
      ...
      <expression id="eN"><![CDATA[<expression>...</expression>]]></expression>
   </cartesian-square>
</Item>
```

You must provide at least two expressions within the 'cartesian-square' node

%XML|JSON%	Specifies whether the response format is XML or JSON. The default is JSON.

targetScope - ***%builder method name%*** - ***%builder method attributes%*** - ***%builder method properties%***	Refer to "4.3.2 Input Item Format" targetScope is required
<cartesian-square>…</cartesian-square>	Node that contain expressions.

Inside the API method, the Cartesian square is converted to a Cartesian product with the same two sets. The list of expressions from the Cartesian square goes to each of the two sets of the Cartesian product.

```
<cartesian-square>
      <expression id="e1"><![CDATA[<expression>...</expression>]]></expression>
      <expression id="e2"><![CDATA[<expression>...</expression>]]></expression>
      ...
      <expression id="eN"><![CDATA[<expression>...</expression>]]></expression>
</cartesian-square>
```

The previous expression is then transparently converted to:

```
   <cartesian-product>
      <set>
         <expression id="e1"><![CDATA[<expression>...</expression>]]></expression>
         <expression id="e2"><![CDATA[<expression>...</expression>]]></expression>
         ...
         <expression id="eN"><![CDATA[<expression>...</expression>]]></expression>
      </set>
      <set>
         <expression id="e1"><![CDATA[<expression>...</expression>]]></expression>
         <expression id="e2"><![CDATA[<expression>...</expression>]]></expression>
         ...
         <expression id="eN"><![CDATA[<expression>...</expression>]]></expression>
      </set>
   </cartesian-product>
```

The rest logic of the API method stays the same.

5.4.2 AML Response Format

The response is a list of corteges. A Cortege is a list of intersecting expressions. If there are no intersections, the result is empty.

5.4.2.1 XML format

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            <cortege>
                <expression id="e1" />
                <expression id="e2" />
```

```
                ...
                <expression id="eN" />
            </cortege>
            ...
            <cortege>
                <expression id="eM1" />
                <expression id="eM2" />
                ...
                <expression id="eMN" />
            </cortege>
        </Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If there are no groups, the result node is empty:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result />
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.4.2.2 JSON format

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
          [
            [
              {
                "id": "e1"
              },
              {
                "id": "e2"
              },
             ...
              {
                "id": "eN"
              }
            ],
            [
              {
                "id": "eE1"
              },
              {
                "id": "eE2"
              },
              ...
              {
                "id": "eEN"
              }
            ],
            [
              {
```

A cortege

An expression

```
            "id": "eM1"
          },
          {
            "id": "eM2"
          },
          ...
          {
            "id": "eMN"
          }
        ]
      ]
    </Result>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If there are no groups, the result node displays an empty JSON array:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>[]</Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.4.3 Examples

- Calculating the intersection of two sets with a single expression in each one

```
<Item type="Method" action="Cfg_GetIntersectingExpressions" responseFormat="XML">
   <targetScope>
      <Item type="Method" action="builder_method_name" id="business_item_id"/>
   </targetScope>
   <cartesian-product>
      <set>
         <expression id="e1">
            <![CDATA[
               <expression>
                  <EQ>
                     <variable id="item_id_color" />
                     <named-constant id="item_id_red" />
                  </EQ>
               </expression>
            ]]>
         </expression>
      </set>
      <set>
         <expression id="e2">
            <![CDATA[
               <expression>
                  <EQ>
                     <variable id="item_id_wheelsize" />
                     <named-constant id="item_id_15inch" />
                  </EQ>
               </expression>
```

```
            ]]>
         </expression>
      </set>
   </cartesian-product>
    </Item>
```

The cfg_GetIntersectingExpressions method confirms that a Scope has valid combinations for ('e1' AND 'e2').

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <cortege>
            <expression id="e1" />
            <expression id="e2" />
         </cortege>
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (if @responseFormat="JSON" is within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
        [
          [
            {
              "id": "e1"
            },
            {
              "id": "e2"
            }
          ]
        ]
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Computing the intersection of two sets with two expressions in each one

```
<Item type="Method" action="Cfg_GetIntersectingExpressions" responseFormat="XML">
   <targetScope>
      <Item type="Method" action="builder_method_name" id="business_item_id"/>
   </targetScope>
   <cartesian-product>
      <set>
         <expression id="e1">
            <![CDATA[
               <expression>
                  <EQ>
                     <variable id="item_id_color" />
```

```
                    <named-constant id="item_id_red" />
                </EQ>
            </expression>
        ]]>
    </expression>
    <expression id="e2">
        <![CDATA[
            <expression>
                <EQ>
                    <variable id="item_id_color" />
                    <named-constant id="item_id_green" />
                </EQ>
            </expression>
        ]]>
    </expression>
</set>
<set>
    <expression id="e3">
        <![CDATA[
            <expression>
                <EQ>
                    <variable id="item_id_wheelsize" />
                    <named-constant id="item_id_15inch" />
                </EQ>
            </expression>
        ]]>
    </expression>
    <expression id="e4">
        <![CDATA[
            <expression>
                <EQ>
                    <variable id="item_id_wheelsize" />
                    <named-constant id="item_id_16inch" />
                </EQ>
            </expression>
        ]]>
    </expression>
</set>
</cartesian-product>
</Item>
```

Let's say the Cfg_GetIntersectingExpressions method confirms that a Scope has the following available combinations for:

1. 'e1' AND 'e3' is a valid combination
2. 'e2' AND 'e3' is a valid combination
3. 'e2' AND 'e4' is a valid combination

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <cortege>
```

```
            <expression id="e1" />
            <expression id="e3" />
         </cortege>
         <cortege>
            <expression id="e2" />
            <expression id="e3" />
         </cortege>
         <cortege>
            <expression id="e2" />
            <expression id="e4" />
         </cortege>
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (if @responseFormat="JSON" is within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
        [
          [
            {
              "id": "e1"
            },
            {
              "id": "e3"
            }
          ],
          [
            {
              "id": "e2"
            },
            {
              "id": "e3"
            }
          ],
          [
            {
              "id": "e2"
            },
            {
              "id": "e4"
            }
          ]
        ]
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Calculating the Cartesian square of three expressions

```
<Item type="Method" action="Cfg_GetIntersectingExpressions" responseFormat="XML">
   <targetScope>
      <Item type="Method" action="builder_method_name" id="business_item_id"/>
   </targetScope>
```

```
   <cartesian-square>
      <expression id="e1">
         <![CDATA[
            <expression>
               <EQ>
                  <variable id="item_id_color" />
                  <named-constant id="item_id_red" />
               </EQ>
            </expression>
         ]]>
      </expression>
      <expression id="e2">
         <![CDATA[
            <expression>
               <EQ>
                  <variable id="item_id_wheelsize" />
                  <named-constant id="item_id_18inch" />
               </EQ>
            </expression>
            ]]>
      </expression>
      <expression id="e3">
        <![CDATA[
           <expression>
             <EQ>
               <variable id="item_id_bodytype" />
                <named-constant id="item_id_sport" />
             </EQ>
           </expression>
         ]]>
      </expression>
   </cartesian- square>
</Item>
```

Let's say the Cfg_GetIntersectingExpressions method confirms that a Scope has the following available combinations for:

1. 'e1' AND 'e2' is a valid combination
2. 'e1' AND 'e3' is a valid combination
3. 'e2' AND 'e3' is a valid combination

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <cortege>
            <expression id="e1" />
            <expression id="e2" />
         </cortege>
         <cortege>
            <expression id="e1" />
            <expression id="e3" />
         </cortege>
```

```
         <cortege>
            <expression id="e2" />
            <expression id="e3" />
         </cortege>
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (if @responseFormat="JSON" within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
        [
          [
            {
              "id": "e1"
            },
            {
              "id": "e2"
            }
          ],
          [
            {
              "id": "e1"
            },
            {
              "id": "e3"
            }
          ],
          [
            {
              "id": "e2"
            },
            {
              "id": "e3"
            }
          ]
        ]
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.5 CfgGetConflicts

The API method described in this section will help you to find the reason why the Scope does not have a solution. Use the CfgGetConflicts internal server method to find and describe all contradictions between existing rules, variable constraints, and custom conditions.

The method returns a list of found conflicts. Each conflict contains a list of conflict sources (rules, variables, missing items, conditions).

5.5.1 AML Request Syntax

The AML request syntax is as follows:

```
<Item type="Method" action="cfg_GetConflicts">
    <targetScope>
        <Item type="Method" action="%builder method name%" %builder method attributes%>
         %builder method properties%
        </Item>
    </targetScope>
    <condition>…</condition>
</Item>
```

targetScope - ***%builder method name%*** - ***%builder method attributes%*** - ***%builder method properties%***	Refer to "4.3.2 Input Item Format" targetScope is required.
<condition>…</condition>	Node that contains the expression to add to the Scope before finding combinations. The expression is included in the list of conflict sources. The Default value is empty.

5.5.2 AML Response Format

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            <Conflict>
                <member source="condition" propositionalForm="...">
                    <![CDATA[<expression>...</expression>]]>
                </member>
                <member source="variable" id="..." propositionalForm="...">
                    <![CDATA[<expression>...</expression>]]>
                </member>
                <member source="rule" id="..." propositionalForm="...">
                    <![CDATA[<expression>...</expression>]]>
                </member>
                <member source="missinginscope" propositionalForm="...">
                    <![CDATA[<expression>...</expression>]]>
                </member>
            </Conflict>
            <Conflict>
                ...
            </Conflict>
            ...
            </Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The <Result> node contains a <Conflict> child node for each found conflict. The <Conflict> node consists of <member> child nodes that represent a certain expression that causes conflict.

The <member> node contains the following:

- source attribute stores the member type: rule, variable, condition or missinginscope.
- propositionalForm attribute holds the text representation of the member's inner expression.
- id attribute stores the corresponding Rule or Variable ID if the "source" attribute equals to "rule" or "variable".
- inner text, which contains the serialized expression of this conflict member.

Conflict source types are follows:

- rule corresponds to a Scope Rule.
- variable corresponds to a Scope Variable.
- condition corresponds to the expression specified as input.
- missinginscope corresponds to an expression containing a single equivalence. An equivalence is "missing" if one or both equivalence parts are not in Scope. The Configurator Services API considers such an equivalence to be False.

5.5.3 Examples

Scope contains the following variables:

- Color: Red, Green, Blue, Black, White
- Style: Wagon, Business, Sport
- Wheel Size: 14", 15", 16", 17"
- Wheel Type: Steel, Aluminum

Scope contains the following rules:

- "Red" is the only color available for "Sport" Style.
- "Business" is only available in the colors "Black" or "White".
- "Wagon" Style is only available for "14"" and "15"" Wheel Size.
- "Sport" Style is only available for "16"" and "17"" Wheel Size.
- "14"" Wheel Size is only available for the "Steel" Wheel Type.
- "17"" Wheel Size is only available for the "Aluminum" Wheel Type.

This scope has the following valid combinations:

- Color=Green, Style=Wagon, WheelSize=14", WheelType=Steel.
- Color=White, Style=Business, WheelSize=15", WheelType=Steel.
- Color=Red, Style=Sport, WheelSize=17", WheelType=Aluminum.
- Valid Scope

Request:

```
<Item type="Method" action="cfg_GetConflicts">
    <targetScope>…</targetScope>
</Item>
```

Response:

Because the scope is solvable, the result is empty.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result />
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Valid Scope – check with condition

Request:

```
<Item type="Method" action="cfg_GetConflicts">
    <targetScope>…</targetScope>
    <condition><![CDATA[
        <expression>
            <EQ>
                <variable id="IDOFCOLOR" />
                <named-constant id="IDOFGREEN" />
            </EQ>
            <EQ>
                <variable id="IDOFSTYLE" />
                <named-constant id="IDOFWAGON" />
            </EQ>
            <EQ>
                <variable id="IDOFWHEELSIZE" />
                <named-constant id="IDOF14INCHES" />
            </EQ>
            <EQ>
                <variable id="IDOFWHEELTYPE" />
                <named-constant id="IDOFSTEEL" />
            </EQ>
        </expression>
    ]]></condition>
</Item>
```

Response:

Because the scope is solvable, the result is empty.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result />
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Simple single conflict

Request:

```
<Item type="Method" action="cfg_GetConflicts">
```

```
    <targetScope>…</targetScope>
    <condition><![CDATA[
        <expression>
            <EQ>
                <variable id="IDOFCOLOR" />
                <named-constant id="IDOFRED" />
            </EQ>
            <EQ>
                <variable id="IDOFSTYLE" />
                <named-constant id="IDOFWAGON" />
            </EQ>
            <EQ>
                <variable id="IDOFWHEELSIZE" />
                <named-constant id="IDOF14INCHES" />
            </EQ>
            <EQ>
                <variable id="IDOFWHEELTYPE" />
                <named-constant id="IDOFSTEEL" />
            </EQ>
        </expression>
    ]]></condition>
</Item>
```

Response:

The following condition contains the conflicting variants “Red” and “Wagon” and the rule which makes their simultaneous usage impossible:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            <Conflict>
                <member source="condition" propositionalForm="..."><![CDATA[
                    <expression>
                        <EQ>
                            <variable id="IDOFCOLOR" />
                            <named-constant id="IDOFRED" />
                        </EQ>
                        <EQ>
                            <variable id="IDOFSTYLE" />
                            <named-constant id="IDOFWAGON" />
                        </EQ>
                    </expression>
                ]]></member>
                <member source="rule" id="1234" propositionalForm="IF Color=Red THEN
Style=Sport">…</member>
        </Conflict>
        </Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Just one value can be set to Variable

Request:

```
<Item type="Method" action="cfg_GetConflicts">
    <targetScope>…</targetScope>
    <condition><![CDATA[
```

```
        <expression>
            <EQ>
                <variable id="IDOFCOLOR" />
                <named-constant id="IDOFRED" />
            </EQ>
            <EQ>
                <variable id="IDOFCOLOR" />
                <named-constant id="IDOFGREEN" />
            </EQ>
        </expression>
    ]]></condition>
</Item>
```

Response:

These two values contradict each other because both of them belong to the "Color" Variable. The Variable cannot contain two values simultaneously.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            <Conflict>
                <member source="condition" propositionalForm="Color=Red AND
Color=Green"><![CDATA[
                    <expression>
                        <EQ>
                            <variable id="IDOFCOLOR" />
                            <named-constant id="IDOFRED" />
                        </EQ>
                        <EQ>
                            <variable id="IDOFCOLOR" />
                            <named-constant id="IDOFGREEN" />
                        </EQ>
                    </expression>
                ]]></member>
                <member source="variable" id="IDOFCOLOR" propositionalForm="EXACTLY-ONE
(Color=Red | Color=Green | Color=Blue | Color=Black | Color = White)">…</member>
            </Conflict>
        </Result>
    </SOAP-ENV:Body>
      </SOAP-ENV:Envelope>
```

- Variable should have value

Request:

```
<Item type="Method" action="cfg_GetConflicts">
    <targetScope>…</targetScope>
    <condition><![CDATA[
        <expression>
            <NOT>
                <EQ>
                    <variable id="IDOFSTYLE" />
                    <named-constant id="IDOFWAGON" />
                </EQ>
            </NOT>
            <NOT>
                <EQ>
```

```
                <variable id="IDOFSTYLE" />
                <named-constant id="IDOBUSINESS" />
            </EQ>
        </NOT>
        <NOT>
            <EQ>
                <variable id="IDOFSTYLE" />
                <named-constant id="IDOFSPORT" />
            </EQ>
        </NOT>
    </expression>
]]></condition>
</Item>
```

Response:
The following three values are the full list of possible values for the Style Variable. The condition says that the Style Variable can't be any of this values. The Variable cannot be empty.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <Conflict>
            <member source="condition" propositionalForm="NOT (Style=Wagon) AND NOT
(Style=Business) AND NOT (Style=Sport)" ><![CDATA[
            <expression>
               <NOT>
                  <EQ>
                     <variable id="IDOFSTYLE" />
                     <named-constant id="IDOFWAGON" />
                  </EQ>
               </NOT>
               <NOT>
                  <EQ>
                     <variable id="IDOFSTYLE" />
                     <named-constant id="IDOBUSINESS" />
                  </EQ>
               </NOT>
               <NOT>
                  <EQ>
                     <variable id="IDOFSTYLE" />
                     <named-constant id="IDOFSPORT" />
                  </EQ>
               </NOT>
            </expression>]]>
            </member>
            <member source="variable" id="IDOFSTYLE" propositionalForm=" EXACTLY-ONE
(Style=Wagon | Style=Business | Style=Sport)">…</member>
         </Conflict>
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Multiple conflicts

Request:

```
<Item type="Method" action="cfg_GetConflicts">
    <targetScope>…</targetScope>
    <condition><![CDATA[
        <expression>
            <EQ>
                <variable id="IDOFCOLOR" />
                <named-constant id="IDOFRED" />
            </EQ>
            <EQ>
                <variable id="IDOFSTYLE" />
                <named-constant id="IDOFWAGON" />
            </EQ>
            <EQ>
                <variable id="IDOFWHEELSIZE" />
                <named-constant id="IDOF17INCHES" />
            </EQ>
            <EQ>
                <variable id="IDOFWHEELTYPE" />
                <named-constant id="IDOFSTEEL" />
            </EQ>
        </expression>
    ]]></condition>
</Item>
```

Response:

This combination contains multiple contradictions. Each of them transforms into a <Conflict>-node.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            <Conflict>
                <member source="condition" propositionalForm="Style=Wagon AND
Color=Red"><![CDATA[
                    <expression>
                        <EQ>
                            <variable id="IDOFCOLOR" />
                            <named-constant id="IDOFRED" />
                        </EQ>
                        <EQ>
                            <variable id="IDOFSTYLE" />
                            <named-constant id="IDOFWAGON" />
                        </EQ>
                    </expression>
                ]]></member>
                <member source="rule" id="1234" propositionalForm="IF Color=Red THEN
Style=Sport">…</member>
            </Conflict>
            <Conflict>
                <member source="condition" propositionalForm='Style=Wagon AND [Wheel
Size]=[17"]”'><![CDATA[
                    <expression>
                        <EQ>
```

```
                        <variable id="IDOFSTYLE" />
                        <named-constant id="IDOFWAGON" />
                    </EQ>
                    <EQ>
                        <variable id="IDOFWHEELSIZE" />
                        <named-constant id="IDOF17INCHES" />
                    </EQ>
                </expression>
            ]]></member>
            <member source="rule" id="5678" propositionalForm='IF Style=Wagon THEN
([Wheel Size]=[14"] OR [Wheel Size]=[15"])'>…</member>
        </Conflict>
        <Conflict>
            <member source="condition" propositionalForm='[Wheel Type]=Steel AND [Wheel
Size]=[17"]'><![CDATA[
                <expression>
                    <EQ>
                        <variable id="IDOFWHEELSIZE" />
                        <named-constant id="IDOF17INCHES" />
                    </EQ>
                    <EQ>
                        <variable id="IDOFWHEELTYPE" />
                        <named-constant id="IDOFSTEEL" />
                    </EQ>
                </expression>
            ]]></member>
            <member source="rule" id="0011" propositionalForm='IF [Wheel Size]=[17"]
THEN [Wheel Type]= Aluminum'>…</member>
        </Conflict>
    </Result>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Unobvious conflicts

  Variables: "Size" { "Big", "Small" } and "Color" { "Black", "White" }

  Rules:

  "*IF (Size=Big) OR (Color=Black) THEN (Size=Big) AND (Color=Black)*",

  "*IF (Size=Small) OR (Color=White) THEN (Size=Small) AND (Color=White)*".

  This will have two valid combinations: "Big","Black" and "Small","White".

  When we try the condition *(Color=White) AND (Color=Black)*, we will get two conflicts instead of one:

  Request:

```
<Item type="Method" action="cfg_GetConflicts">
    <targetScope>…</targetScope>
    <condition><![CDATA[
        <expression>
            <EQ>
                <variable id="IDOFSIZE" />
                <named-constant id="IDOFBIG" />
            </EQ>
            <EQ>
                <variable id="IDOFSIZE" />
                <named-constant id="IDOFSMALL" />
```

```
            </EQ>
        </expression>
    ]]></condition>
</Item>
```

Response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            <Conflict><!-- This conflict is obvious and expected. -->
                <member source="condition" propositionalForm="Size=Big AND
Size=Small"><![CDATA[
                    <expression>
                    <EQ>
                        <variable id="IDOFSIZE" />
                        <named-constant id="IDOFBIG" />
                    </EQ>
                    <EQ>
                        <variable id="IDOFSIZE" />
                        <named-constant id="IDOFSMALL" />
                    </EQ>
                    </expression>
                ]]></member>
                <member source="variable" id="IDOFSIZE" propositionalForm="EXACTLY-ONE
(Size=Big | Size = Small)">…</member>
            </Conflict>
            <Conflict>
                <member source="condition" propositionalForm="Size=Big and
Size=Small"><![CDATA[
                    <expression>
                    <EQ>
                        <variable id="IDOFSIZE" />
                        <named-constant id="IDOFBIG" />
                    </EQ>
                    <EQ>
                        <variable id="IDOFSIZE" />
                        <named-constant id="IDOFSMALL" />
                    </EQ>
                    </expression>
                ]]></member>
                <member source="variable" id="IDOCOLOR" propositionalForm="EXACTLY-ONE
                (Color=Black | Color=White)">…</member>
                <member source="rule" id="XXYYZZ" propositionalForm="IF (Size=Big) OR
(Color=Black) THEN (Size=Big) AND (Color=Black)">…</member>
                <member source="rule" id="UUVVWW" propositionalForm="IF (Size=Small) OR
(Color=White) THEN (Size=Small) AND (Color=White)">…</member>
            </Conflict>
        </Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The explanation of the last conflict is as follows:

- Value "Big" and Rule "IF (Size=Big) OR (Color=Black) THEN (Size=Big) AND (Color=Black)" together means that the "Black" value must be selected.
- Value "Small" and Rule "IF (Size=Small) OR (Color=White) THEN (Size=Small) AND (Color=White)" together means that "White" value must be selected.
- The colors Black and White cannot be chosen at the same time "EXACTLY-ONE (Color=Black | Color=White)". Only one value can be set for the variable.

- Using a Boolean expression as a condition

It is possible to use a complex Boolean expression as an input parameter:

Request:

```
<Item type="Method" action="cfg_GetConflicts">
<targetScope>…</targetScope>
    <condition><![CDATA[<expression>
        <OR>
            <AND>
                    <EQ>
                        <variable id="IDOFSIZE" />
                        <named-constant id="IDOFBIG" />
                    </EQ>
                    <EQ>
                        <variable id="IDOFCOLOR" />
                        <named-constant id="IDOFWHITE" />
                    </EQ>
            </AND>
            <AND>
                    <EQ>
                        <variable id="IDOFSIZE" />
                        <named-constant id="IDOFSMALL" />
                    </EQ>
                    <EQ>
                        <variable id="IDOFCOLOR" />
                        <named-constant id="IDOFBLACK" />
                    </EQ>
            </AND>
        </OR>
    </expression>]]></condition>
</Item>
```

This condition describes the disjunction of two invalid combinations.

Response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <Result>
   <Conflict>
    <member source="condition" propositionalForm="(Size=Big AND Color=White) OR
(Size=Small AND Color=Black)"><![CDATA[<expression>
     <OR>
      <AND>
```

```
        <EQ>
         <variable id="IDOFSIZE" />
         <named-constant id="IDOFBIG" />
        </EQ>
        <EQ>
         <variable id="IDOFCOLOR" />
         <named-constant id="IDOFWHITE" />
        </EQ>
       </AND>
       <AND>
        <EQ>
         <variable id="IDOFSIZE" />
         <named-constant id="IDOFSMALL" />
        </EQ>
        <EQ>
         <variable id="IDOFCOLOR" />
         <named-constant id="IDOFBLACK" />
        </EQ>
       </AND>
      </OR>
    </expression>]]></member>
    <member source="variable" id="IDOFSIZE" propositionalForm="EXACTLY-ONE
(Size=Big | Size = Small)">…</member>
    <member source="variable" id="IDOFCOLOR" propositionalForm="EXACTLY-ONE
(Color=Black | Color = White)">…</member>
    <member source="rule" id="XXYYZZ" propositionalForm="IF (Size=Big) OR
(Color=Black) THEN (Size=Big) AND (Color=Black)">…</member>
   </Conflict>
   <Conflict>
    <member source="condition" propositionalForm="(Size=Big AND Color=White) OR
(Size=Small AND Color=Black)"><![CDATA[<expression>
     <OR>
      <AND>
       <EQ>
        <variable id="IDOFSIZE" />
        <named-constant id="IDOFBIG" />
       </EQ>
       <EQ>
        <variable id="IDOFCOLOR" />
        <named-constant id="IDOFWHITE" />
       </EQ>
      </AND>
      <AND>
       <EQ>
        <variable id="IDOFSIZE" />
        <named-constant id="IDOFSMALL" />
       </EQ>
       <EQ>
        <variable id="IDOFCOLOR" />
        <named-constant id="IDOFBLACK" />
       </EQ>
      </AND>
```

```
     </OR>
    </expression>]]></member>
    <member source="variable" id="IDOFSIZE" propositionalForm="EXACTLY-ONE
(Size=Big | Size = Small)">…</member>
    <member source="variable" id="IDOFCOLOR" propositionalForm="EXACTLY-ONE
(Color=Black | Color = White)">…</member>
    <member source="rule" id="UUVVWW" propositionalForm="IF (Size=Small) OR
(Color=White) THEN (Size=Small) AND (Color=White)">…</member>
   </Conflict>
   <Conflict>
    <member source="condition" propositionalForm="(Size=Big AND Color=White) OR
(Size=Small AND Color=Black)"><![CDATA[<expression>
     <OR>
      <AND>
       <EQ>
        <variable id="IDOFSIZE" />
        <named-constant id="IDOFBIG" />
       </EQ>
       <EQ>
        <variable id="IDOFCOLOR" />
        <named-constant id="IDOFWHITE" />
       </EQ>
      </AND>
      <AND>
       <EQ>
        <variable id="IDOFSIZE" />
        <named-constant id="IDOFSMALL" />
       </EQ>
       <EQ>
        <variable id="IDOFCOLOR" />
        <named-constant id="IDOFBLACK" />
       </EQ>
      </AND>
     </OR>
    </expression>]]></member>
    <member source="variable" id="IDOFSIZE" propositionalForm="EXACTLY-ONE
(Size=Big | Size = Small)">…</member>
    <member source="rule" id="XXYYZZ" propositionalForm="IF (Size=Big) OR
(Color=Black) THEN (Size=Big) AND (Color=Black)">…</member>
    <member source="rule" id="UUVVWW" propositionalForm="IF (Size=Small) OR
(Color=White) THEN (Size=Small) AND (Color=White)">…</member>
   </Conflict>
   <Conflict>
    <member source="condition" propositionalForm="(Size=Big AND Color=White) OR
(Size=Small AND Color=Black)"><![CDATA[<expression>
     <OR>
      <AND>
       <EQ>
        <variable id="IDOFSIZE" />
        <named-constant id="IDOFBIG" />
       </EQ>
       <EQ>
```

```
        <variable id="IDOFCOLOR" />
        <named-constant id="IDOFWHITE" />
       </EQ>
      </AND>
      <AND>
       <EQ>
        <variable id="IDOFSIZE" />
        <named-constant id="IDOFSMALL" />
       </EQ>
       <EQ>
        <variable id="IDOFCOLOR" />
        <named-constant id="IDOFBLACK" />
       </EQ>
      </AND>
     </OR>
    </expression>]]></member>
    <member source="variable" id="IDOFCOLOR" propositionalForm="EXACTLY-ONE
(Color=Black | Color = White)">…</member>
    <member source="rule" id="XXYYZZ" propositionalForm="IF (Size=Big) OR
(Color=Black) THEN (Size=Big) AND (Color=Black)">…</member>
    <member source="rule" id="UUVVWW" propositionalForm="IF (Size=Small) OR
(Color=White) THEN (Size=Small) AND (Color=White)">…</member>
   </Conflict>
  </Result>
```

- Missing terms

It is possible for some terms to be missing from the scope.

The request contains an expression that specifies a color, which is out of scope:

```
<Item type="Method" action="cfg_GetConflicts">
    <targetScope>…</targetScope>
    <condition><![CDATA[<expression>
        <EQ>
           <variable id="IDOFCOLOR" />
           <named-constant id="YELLOW" />
        </EQ>
    </expression>]]></condition>
</Item>
```

If this term causes any conflict, the corresponding <member> nodes reflect this fact:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <Result>
   <Conflict>
    <member source="condition"
propositionalForm="Color=Yellow"><![CDATA[<expression>
     <EQ>
      <variable id="IDOFCOLOR" />
      <named-constant id="YELLOW" />
      </EQ>
     </expression>]]>
```

```
    </member>
    <member source="missinginscope"
propositionalForm="NOT(Color=Yellow)">…</member>
   </Conflict>
  </Result>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Scope with no valid combinations

Scope with:

  - Variable "Color" { "Black", "White" }.
  - Variable "Size" { "Small" }.
  - Rule "IF (Color=Black) OR (Color=White) THEN NOT (Size=Small)".

The Scope is invalid because the Variable cannot be assigned.

Request:

```
<Item type="Method" action="cfg_GetConflicts">
       <targetScope>…</targetScope>
   <condition />
</Item>
```

Response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <Conflict>
            <member source="variable" id="IDOFCOLOR" propositionalForm="EXACTLY-ONE
(Color=Black | Color = White)">…</member>
            <member source="rule" id="UUVVWW" propositionalForm="IF (Color=Black) OR
(Color=White) THEN NOT (Size=Small)">…</member>
         </Conflict>
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Input condition expressions are not necessary to cause conflicts.

5.6 Cfg_GetExpressionTruthTable

The Cfg_GetExpressionTruthTable method is very helpful for debugging purposes. Analyzing and resolving the process can reveal the direct contradictions between different parts of the scope, which may not be so obvious.

- The method builds a truth table for the specified expression.
- The method works without the Scope being specified.
- The method auto generates Scope inside the method implementation.

- For each equivalence from the expression the new Variable is added to the auto generated Scope.
- All Variables in the auto generated Scope have two possible namedConstants: 0, 1.
- The Expression specified in the request translates to new Variables.

Example: for the expression (if Color=White then Color=Black) and (if Color=Black then Color=White)

AutoGeneratedScope

[Color=White]

1

0

[Color=Black]

1

0

Translated expression:

(IF [Color=White] = 1 THEN [Color=Black] = 1) AND (IF [Color=Black] = 1 THEN [Color=White] = 1)

The Cfg_GetExpressionTruthTable method finds and returns all valid combinations for the new AutoGeneratedScope using the translated expression as a condition.

5.6.1 AML Request Syntax

```
<Item type="Method" action="cfg_GetExpressionTruthTable" responseFormat="%XML|JSON%">
    <condition>
        <![CDATA[
        <expression>...</expression>
        ]]>
    </condition>
</Item>
```

%XML|JSON%	Specifies whether the response format is XML or JSON. The default is JSON.
<condition>…</condition>	A Node that contains the expression.The Default value is empty.

5.6.2 AML Response Format

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <truth-table-meta>
            <terms>
               <term order="0"><![CDATA[<expression>…</expression>]]></term>
               <term order="1">…</term>
               …
               </terms>
         </truth-table-meta>
```

```
            <truth-table>
                <combination>
                    <value>1</value>
                    <value>0</value>
                    …
                    </combination>
                …
                </truth-table>
        </Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The method builds a response in the same way as GetValidCombinations.

The `<truth-table-meta>` consists of `<terms>`, where the `<term>` is an expression with the equivalence inside. The term itself is a Boolean variable with values of either 0 or 1. The `<truth-table>` consists of combinations, where each `<combination>` represents the values of terms that lead the initial condition to be true.

The JSON response (@responseFormat="JSON" within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            {
               "truth-table-meta": {
                  terms: [
                     { expression: "<eq>…</eq>" },
                     …
                     { expression: "<eq>…</eq>" },
                  ]
               },
               "truth-table": [
                     [.., …, ..],
                     …
                     [.., …, ..]
               ]
            }
        </Result>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.6.3 Examples

- Build truth table

Build a truth table for EXACTLY-ONE(TIRES_TYPE=[SC 3], TIRES_TYPE=[PC 5], TIRES_TYPE=[SC 5]).

There are three rows in the truth table:

TIRES_TYPE=[SC 3]	TIRES_TYPE=[PC 5],	TIRES_TYPE=[SC 5]
1	0	0
0	1	0
0	0	1

Request:

```
<Item type="Method" action="cfg_GetExpressionTruthTable">
  <condition><![CDATA[<expression>
  <exactly-one>
    <eq>
      <variable id="TIRES_TYPE" />
      <named-constant id="SC 3" />
    </eq>
    <eq>
      <variable id="TIRES_TYPE" />
      <named-constant id="PC 5" />
    </eq>
    <eq>
      <variable id="TIRES_TYPE" />
      <named-constant id="SC 5" />
    </eq>
  </exactly-one>
</expression>]]>
  </condition>
</Item>
```

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <Result>
            <truth-table-meta>
                <terms>
                    <term order="0">
                        <![CDATA[<expression>
                            <eq><variable id="TIRES_TYPE" /><named-constant id="SC 3" /></eq>
                        </expression>]]>
                    </term><term order="1">
                        <![CDATA[<expression>
                            <eq><variable id="TIRES_TYPE" /><named-constant id="PC 5" /></eq>
                        </expression>]]>
                    </term><term order="2">
                        <![CDATA[<expression>
                            <eq><variable id="TIRES_TYPE" /><named-constant id="SC 5" /></eq>
                        </expression>]]>
                    </term>
                </terms>
            </truth-table-meta>
            <truth-table>
                <combination>
```

```
                <value>1</value>
                <value>0</value>
                <value>0</value>
            </combination>
            <combination>
                <value>0</value>
                <value>1</value>
                <value>0</value>
            </combination>
            <combination>
                <value>0</value>
                <value>0</value>
                <value>1</value>
            </combination>
        </truth-table>
    </Result>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (@responseFormat="JSON" within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         {
            "truth-table-meta": {
                  terms: [
                        { expression: "<eq><variable id=\"TIRES_TYPE\" /><named-
constant id=\"SC 3\" /></eq>" },
                        { expression: "<eq><variable id=\"TIRES_TYPE\" /><named-
constant id=\"SC 5\" /></eq>" },
                        { expression: "<eq><variable id=\"TIRES_TYPE\" /><named-
constant id=\"PC 5\" /></eq>" },
                  ]
            },
            "truth-table": [
            [1, 0, 0],
            [0, 1, 0],
            [0, 0, 1]
            ]
      }
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.7 Extra - How To

5.7.1 Get Unreachable Combinations

5.7.1.1 Overview

This section describes how to get a list of unreachable combinations. A combination is unreachable if it is valid in the current Scope, but does not satisfy the specified expression. To resolve this task, use the GetValidCombinations method.

5.7.1.2 Prepare Expression

Task: Find unreachable combinations using the expression `<expression>e_1</expression>`.

Prepared expression: `<expression><not>e_1</not></expression>`

Task: Need to find unreachable combinations using a list of expressions:

```
<expression>e_1</expression>
<expression>e_2</expression>
<expression>e_3</expression>.
```

Prepared expression:

```
<expression>
    <not>
        <or>
            e_1
            e_2
            e_3
        </or>
    </not>
</expression>
```

A single expression

```
<expression>
   ...
</expression>
```

expression content is moving to

Unreachable combinations condition node should look like:

```
<condition>
  <![CDATA[
    <expression>
      <NOT>
        ...
      </NOT>
    </expression>
   ]]>
</condition>
```

Several separate expressions

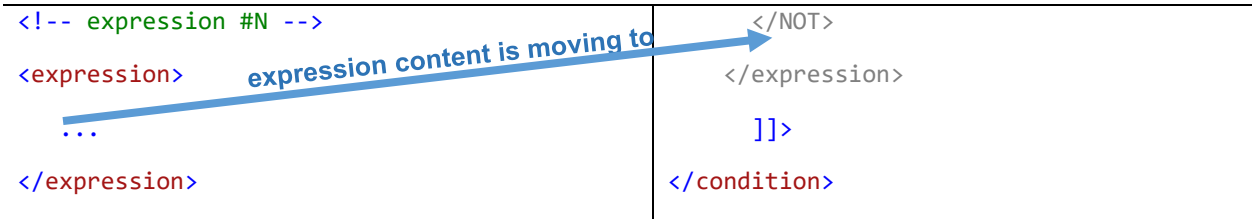
```
<!-- expression #1 -->
<expression>
   ...
</expression>
<!-- expression #2 -->
<expression>
   ...
</expression>
```

expression content is moving to

expression content is moving to

Unreachable combinations condition node should look like:

```
<condition>
  <![CDATA[
    <expression>
      <NOT>
        <OR>
          <!-- expression content #1 -->
          ...
          <!-- expression content #2 -->
          ...
          <!-- expression content #N -->
          ...
        </OR>
```



5.7.1.3 Response Explanation

The GetValidCombinations method returns combinations that aren't covered by any of the initial expressions. If the GetValidCombinations method does not return any combinations (an empty result), it means all valid combinations are "reachable" and are therefore covered by the list of initial expressions.

5.7.1.4 Examples

- Unreachable combinations exist

The Scope contains two variables: 'Color' and 'Engine type'. The scope has no rules. You can set the 'Color' variable to either 'Red' or 'Green' values. You can set the 'Engine type' variable to either 'Diesel' or 'Gasoline' values. Since there are no rules, it means there are four available valid combinations: Red & Diesel, Red & Gasoline, Green & Diesel, Green & Gasoline.

You need to find all valid combinations that are not covered by the following expressions:

Verbal notation	Expression content

'Color' is 'Red'	`<EQ>`
	`<variable id="Color" />`
	`<named-constant id="Red" />`
	`</EQ>`

The request:

```
<Item type="Method" action="cfg_GetValidCombinations" responseFormat="XML">
   <targetScope>
      <Item type="Method" action="…" id="…" />
   </targetScope>
   <condition>
      <![CDATA[
         <expression>
            <NOT>
                  <EQ>
                     <variable id="Color" />
                     <named-constant id="Red" />
                  </EQ>
            </NOT>
         </expression>
      ]]>
   </condition>
</Item>
```

The response contains two combinations: Green & Diesel and Green & Gasoline. Because they aren't covered by the initial expression, they are referred to as unreachable combinations.

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <combinations-meta>
            <variables>
               <variable id="Color" order="0" />
               <variable id="Engine type" order="1" />
            </variables>
            <values>
               <value type="NamedConstant" order="0">Green</value>
               <value type="NamedConstant" order="1">Diesel</value>
               <value type="NamedConstant" order="2">Gasoline</value>
            </values>
         </combinations-meta>
         <combinations>
            <combination>
               <value>0</value>
               <value>1</value>
            </combination>
            <combination>
               <value>0</value>
               <value>2</value>
            </combination>
         </combinations>
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (if @responseFormat="JSON" is within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
       {
          "combinations-meta": {
             "variables": {
                "Color": 0,
                "Engine type": 1
             },
             "values": [{
                "type": "NamedConstant",
                "id": "Green"
             },
             {
                "type": "NamedConstant",
                "id": "Diesel"
             },
             {
                "type": "NamedConstant",
                "id": "Gasoline"
```

```
            }]
         },
         "combinations": [[0,1],[0,2]]
            }
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Unreachable combinations don't exist

The Scope contains two variables: 'Color' and 'Engine type'. The scope has no rules. You can set the 'Color' variable to either 'Red' or 'Green' values. You can set the 'Engine type' variable to either 'Diesel' or 'Gasoline' values. Since there are no rules, it means there are four available valid combinations: Red & Diesel, Red & Gasoline, Green & Diesel, Green & Gasoline.

You need to confirm that all valid combinations are covered by the following expressions:

Verbal notation	Expression content

'Color' is 'Red'	`<EQ>`
	`<variable id="Color" />`
	`<named-constant id="Red" />`
	`</EQ>`

'Color' is 'Green'	`<EQ>`
	`<variable id="Color" />`
	`<named-constant id="Green" />`
	`</EQ>`

The request is:

```
<Item type="Method" action="cfg_GetValidCombinations" responseFormat="XML">
   <targetScope>
      <Item type="Method" action="…" id="…">
   </targetScope>
   <condition>
      <![CDATA[
         <expression>
            <NOT>
               <OR>
                  <EQ>
                     <variable id="Color" />
                     <named-constant id="Red" />
                  </EQ>
                  <EQ>
                     <variable id="Color" />
                     <named-constant id="Green" />
                  </EQ>
               </OR>
            </NOT>
         </expression>
      ]]>
```

```
   </condition>
</Item>
```

Since all valid combinations have the 'Color' set to 'Red' or 'Green', all valid combinations are reachable and the response is empty.

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result />
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (if @responseFormat="JSON" is within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>{}</Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.7.2 Get Unreachable Expressions

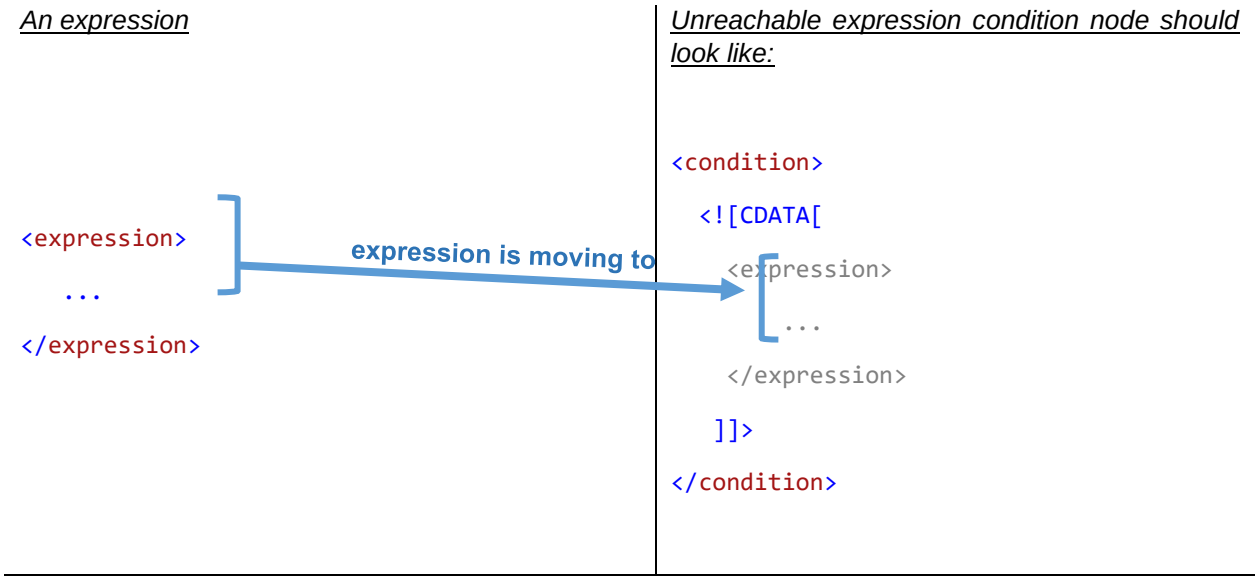
5.7.2.1 Overview

This section describes how to verify that an expression is unreachable. An expression is unreachable if there are no valid combinations in the current Scope using the specified expression. To resolve this task, use the GetValidCombinations method.

5.7.2.2 Prepare Expression

There is no specific format for the 'condition' node of the GetValidCombinations method request.

You must put your expression inside the 'condition' node. The following diagram shows how to create a proper 'condition' node.



5.7.2.3 Response Explanation

If the GetValidCombinations method response contains at least one valid combination, it means specified expression is reachable. If the GetValidCombinations method response is empty, it means the specified expression is unreachable.

5.7.2.4 Examples

- Unreachable expression is detected

The Scope contains two variables: 'Color' and 'Engine type'. The scope has no rules. You can set the 'Color' variable to either 'Red' or 'Green' values. You can set the 'Engine type' variable to either 'Diesel' or 'Gasoline' values. Since there are no rules, it means there are four available valid combinations: Red & Diesel, Red & Gasoline, Green & Diesel, Green & Gasoline.

You need to verify that the following expression is reachable:

Verbal notation	Expression
'Color' is 'Red' AND 'Color' is 'Green'	<expression> <AND> <EQ> <variable id="Color" /> <named-constant id="Red" /> </EQ> <EQ> <variable id="Color" /> <named-constant id="Green" /> </EQ> </AND> </expression>

The request:

```
<Item type="Method" action="cfg_GetValidCombinations" fetch="1" responseFormat="XML">
   <targetScope>
      <Item type="Method" action="…" id="…" />
   </targetScope>
   <condition>
      <![CDATA[
         <expression>
            <AND>
               <EQ>
                  <variable id="Color" />
                  <named-constant id="Red" />
               </EQ>
               <EQ>
```

```
                <variable id="Color" />
                <named-constant id="Green" />
              </EQ>
            </AND>
          </expression>
      ]]>
   </condition>
</Item>
```

The 'Color' variable cannot be set to 'Red' and 'Green' values at the same time. This means that the response will be empty. The passed expression is therefore an unreachable one because it doesn't lead to the availability of at least one valid combination.

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result />
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (if @responseFormat="JSON" is contained within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>{}</Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Unreachable expression is not detected

The scope contains two variables: 'Color' and 'Engine type'. The scope has no rules. You can set the 'Color' variable to either 'Red' or 'Green' values. You can set the 'Engine type' variable to either 'Diesel' or 'Gasoline' values. Since there are no rules, it means there are four available valid combinations: Red & Diesel, Red & Gasoline, Green & Diesel, Green & Gasoline.

You need to verify that the following expression is reachable:

Verbal notation	Expression

'Engine type' is 'Diesel'
 	
OR
 	
'Engine type' is 'Gasoline'
	`<expression>`
	`  <OR>`
	`  <EQ>`
	`  <variable id="Engine type" />`
	`  <named-constant id="Diesel" />`
	`  </EQ>`
	`  <EQ>`
	`  <variable id="Engine type" />`
	`  <named-constant id="Gasoline" />`
	`  </EQ>`
	`  </OR>`
	`</expression>`

The request:

```
<Item type="Method" action="cfg_GetValidCombinations" fetch="1" responseFormat="XML">
   <targetScope>
      <Item type="Method" action="…" id="…" />
   </targetScope>
   <condition>
      <![CDATA[
         <expression>
            <OR>
               <EQ>
                  <variable id="Engine type" />
                  <named-constant id="Diesel" />
               </EQ>
               <EQ>
                  <variable id="Engine type" />
                  <named-constant id="Gasoline" />
               </EQ>
            </OR>
         </expression>
      ]]>
   </condition>
</Item>
```

Since you can set the 'Engine type' variable to either 'Diesel' or 'Gasoline' for a particular combination, the response will contain several valid combinations. This means that the passed expression is reachable, because it leads to the availability of at least one valid combination.

The XML response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
         <combinations-meta>
            <variables>
               <variable id="Color" order="0" />
               <variable id="Engine type" order="1" />
            </variables>
            <values>
               <value type="NamedConstant" order="0">Red</value>
               <value type="NamedConstant" order="1">Green</value>
               <value type="NamedConstant" order="2">Diesel</value>
               <value type="NamedConstant" order="3">Gasoline</value>
            </values>
         </combinations-meta>
         <combinations>
            <combination>
               <value>0</value>
               <value>1</value>
            </combination>
            <combination>
               <value>1</value>
               <value>2</value>
            </combination>
            <combination>
               <value>0</value>
```

```
            <value>3</value>
         </combination>
         <combination>
            <value>1</value>
            <value>3</value>
         </combination>
      </combinations>
   </Result>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The JSON response (if @responseFormat="JSON" is contained within the AML request):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Result>
       {
          "combinations-meta": {
             "variables": {
                "Color": 0,
                "Engine type": 1
             },
             "values": [{
                "type": "NamedConstant",
                "id": "Red"
             },
             {
                "type": "NamedConstant",
                "id": "Green"
             },
             {
                "type": "NamedConstant",
                "id": "Diesel"
             },
             {
                "type": "NamedConstant",
                "id": "Gasoline"
             }]
          },
          "combinations": [[0,2],
          [0,3],
          [1,2],
          [1,3]]
       }
      </Result>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Note: In order to verify the reachability of an expression, finding just a single valid combination is sufficient. It is recommended to use the "fetch='1'" attribute for the AML request to cfg_getValidCombination.

6 Control API

There are two UI Controls available in Configurator Services: RuleEditor and VariantsTree. The following sections describe the APIs for each.

6.1 RuleEditor

The RuleEditor control enables users to input text using an "input group template". You can either enter text manually or by using the intellisense menu. You can also customize the view for individual groups or within a group for specific group types.

6.1.1 Constructor

The Constructor supports one composite parameter. The following table describes its properties.

Name	Type	Description
connectId	String	Mandatory parameter. It is the ID of the DOM element container for VariantsTree control. This element should exist in the DOM document. The VariantsTree DOM is placed as a child into the container. The default is 'variantsTreeContainer'.
contextMenuEnabled	Boolean	Optional parameter. Turns context menu support on/off (widget creation, events tracking). The default is False.
isEditable	Boolean	Optional parameter. Turns text editing for the control on/off. The default is False.

Name	Type	Description

template	Object	Optional composite parameter.
		It must include a 'template' array property containing the group names sequence and the parameters for each group from 'template' keyed with group names.
		Example:
		{
		template: ['RuleConditionGroup'],
		templateGroups: {
		'RuleConditionGroup': {
		type: 'grammar',
		grammarFile: 'RuleEditor/RuleGrammar.txt',
		title: 'Rule condition expression',
		},
		}
		}

6.1.2 Public Fields

Name	Type	Description
containerNode	DOM Node	A reference to the DOM Node used as a container for the control DOM.
domNode	DOM Node	A reference to the DOM Node that corresponds to the control.
contextMenu	Object, MenuWidget	A reference to the menu widget used for the context menu.
intelliSenseMenu	Object, MenuWidget	A reference to the menu widget used for the intellisense menu.

6.1.3 Public Methods

This section describes the public methods.

6.1.3.1 setInputTemplate

Use this method to create a new editor input template. The startNewInput method needs to be called before you can start working with the editor using this template method.

Input parameters

Name	Type	Description
inputTemplate	Object	A template that contains a list of groups used as the 'template' array property as well as settings for each particular group.

6.1.3.2 getInputTemplate

Use this method to get the current editor input template.

Return value

Object. Template descriptor or null.

6.1.3.3 setEditState

Switch edit state for the control.

Input parameters

Name	Type	Description
isEditable	Boolean	The applied value defines how the editor content can be modified.

6.1.3.4 isEditable

Get edit state of the control. It determines whether or not content is editable.

Return value

Boolean

6.1.3.5 toggleCssClasses

This method enables toggling between DOM Node CSS classes.

Input parameters

Name	Type	Description
cssClassNames	String | Array of Strings	Class names that can be added or removed from the Node. You can pass several class names as a string parameter by separating them using the 'space' character.
turnStyleOn	Boolean	If true, classes are added to the Node. If false, classes are removed.
targetNode	DOM Node, Nullable	Node that can be modified.

6.1.3.6 isInputValid

Determines whether or not the values from all the input groups passed validation.

Return value

Boolean

6.1.3.7 isInputStarted

Determines whether or not the template was set and all input groups were created.

Return value

Boolean

6.1.3.8 focus

Places focus into the active or first input group.

6.1.3.9 focusgroup

Places focus into a specific input group.

Input parameters

Name	Type	Description
groupName	String	Group name from template.

6.1.3.10 focusFirstEditableGroup

Places focus on the first input group, which supports user input.

6.1.3.11 undoChange

Undoes the previous editor value change.

6.1.3.12 redoChange

Re-does the change that was undone.

6.1.3.13 getCurrentState

Gets information about the current editor state.

Return value

Object with editor state information.

6.1.3.14 resetChanges

Resets all changes. The Editor returns to its initial values.

6.1.3.15 setStartState

Sets up the editor according to state settings. The current changes queue is dropped.

Input parameters

Name	Type	Description
startState	Object	State information for editor settings restoration.

6.1.3.16 getValue

Gets the current editor value.

Return value

Object with values from all input groups. The key is the group name from the template.

6.1.3.17 getStringValue

Gets the editor value as string.

Return value

String. Concatenation of all input group values.

6.1.3.18 setValue

Sets the editor value.

Input parameters

Name	Type	Description
inputData	Object	Value collection for editor input groups. Key = group name from template.

6.1.3.19 getErrorString

Gets value validation error as a string.

Return value

String. Concatenation of validation errors from all input groups.

6.1.3.20 isEditorFocused

Determines which editor control has focus.

Return value

Boolean

6.1.3.21 isMenuFocused

Determines which target menu control has focus.

Input parameters

Name	Type	Description
targetMenu	Object, MenuWidget	Menu widget used in the editor.

Return value

Boolean

6.1.3.22 isValueModified

Identifies whether or not the initial editor value was modified.

Return value

Boolean

6.1.3.23 registerShortcut

Registers the shortcut for the editor.

Input parameters

Name	Type	Description
keyCode	Number	Button key code.
isCtrl	Boolean	Specifies whether or not the Ctrl key is pressed.
isShift	Boolean	Specifies whether or not the Shift key is pressed.
callbackHandler	Function	Method that is called when the shortcut is activated.

6.1.3.24 getGroupIndex

Gets the index for a particular input group.

Input parameters

Name	Type	Description
targetGroup	Object, InputGroup	Registered editor input group.

6.1.3.25 getGroupByIndex

Gets a particular input group by index.

Input parameters

Name	Type	Description
groupIndex	Number	Group index from template.

Return value

InputGroup

6.1.3.26 getGroupByDomNode

Gets particular input group using the related DOM Node.

Input parameters

Name	Type	Description
targetDomNode	DOM Node	DOM Node belonging to the registered editor input group.

Return value

InputGroup

6.1.3.27 getGroupName

Gets the name of a specific input group.

Input parameters

Name	Type	Description
targetGroup	Object, InputGroup	Registered editor input group.

Return value

String. Group name from template.

6.1.3.28 getGroupByName

Gets registered editor input group by name from the template.

Input parameters

Name	Type	Description
groupName	String	Group name from active editor template.

Return value

InputGroup

6.1.3.29 getGroupsCount

Counts the editor input groups.

Return value

Number

6.1.3.30 getNextGroup

Gets the next input group relative to the target group. If the target group is last, then the first group is returned.

Input parameters

Name	Type	Description
targetGroup	Object, InputGroup	Group relative to which next group is searched.

Return value

InputGroup

6.1.3.31 getPrevGroup

Gets the previous input group relative to the target group. If the target group is last, then the first group is returned.

Input parameters

Name	Type	Description
targetGroup	Object, InputGroup	Group relatively to which previous group is searched.

Return value

InputGroup

6.1.3.32 getGroupDomNode

Gets the previous input group relative to the target group. If target group is last, then the first group is returned.

Input parameters

Name	Type	Description
targetGroup	Object,InputGroup | String	Existing editor input group or it id.

Return value

InputGroup

6.1.3.33 startNewInput

Creates editor input groups based on the current template. All previous input groups are removed.

6.1.3.34 showIntelliSense

Shows the intellisense menu for a particular input group.

Input parameters

Name	Type	Description
targetGroup	Object, InputGroup	Existing editor input group.
allOptions	Boolean	If true, all available group values appear. If false, proposed values are limited based on the current group value.

6.1.3.35 hideIntelliSenseMenu

Hides active intellisense menu.

6.1.4 Events

Events described in this section are based on the Eventable core module. You can assign Handlers for these events using the **addEventListener** method.

6.1.4.1 onFocus

Event fires when the editor control has focus.

6.1.4.2 onBlur

Event fires when the focus moves away from the editor control.

6.1.4.3 onStateChanged

Event fires after group values are changed or the group loses focus.

6.1.4.4 onRender

Event fires after the editor creates DOM for input groups.

6.1.4.5 onValueChanged

Event fires after editor creates DOM for input groups.

6.1.4.6 onGroupValueEntered

Event fires if the group value changed and passed validation.

Input parameters

Name	Type	Description
targetGroup	Object, InputGroup	Affected editor input group.
approvedValue	String	Applied value, which passed validation.

6.1.5 Context Menu Events

The events described in this section belong to the dojo context menu widget. Use the dojo.connect and dojo.on modules to attach handlers for them.

6.1.5.1.1 onMenuInit

Event raised before context menu creation. Use it to create a list of required menu items.

Input parameters

Name	Type	Description
menuEvent	Event	Corresponding native context menu event.

Name	Type	Description
targetGroup	Object, InputGroup	Target editor input group that the context menu is created for.
targetGroupName	String	Input group name from template.

Return value

Array of menu items descriptors.

6.1.5.1.2 onMenuItemClick

Event raised when the user clicks on the editor context menu item.

Input parameters

Name	Type	Description
commandId	String	ID of selected menu item.
contextData	Any	Context information for menu.

Return value

Array of menu items descriptors.

6.2 VariantsTree

The VariantsTree Control enables users to create and interact with a layered view. The view contains the following default layers:

- TreeVisualization
- GroupVisualization

The view is scalable. Context menus are supported.

6.2.1 Constructor

The Constructor supports one composite parameter that can contain the properties described in the following table:

Name	Type	Description
connectId	String	Mandatory property. It contains the DOM element container ID for the VariantsTree control. The Element should exist in the document. The VariantsTree DOM will be placed as a child into container. The default is 'variantsTreeContainer'.

layerVisualizationParameters	Object	Optional composite parameter.
		It contains a collection of additional visual settings for view layers, which can be loaded into view. Collection key = layer id.
		Example:
		{
		'groups': { labelWidth: 200 }
		}

6.2.2 Public Fields

Name	Type	Description
layeredView	Object, MultiLayeredView	Reference to MultiLayeredView control. Most parts of the VariantsTree functionality is based on this core control. It is initialized in the constructor.

6.2.3 Public Methods

This section describes the public methods.

6.2.3.1 loadTree

The loadtree method creates TreeLayer and GroupLayer instances, loads them into the control and then centers the view. The TreeLayer instance is initialized with the treeData input parameter. TreeLayer gets the 'variantsTree' ID and GroupLayer gets the 'groups' ID. The Method can be overridden in order to change the list of loaded layers.

Input parameters

Name	Type	Description
treeData	Object	Hierarchical object data structure with a single root, where the next level children are placed into property children.

6.2.3.2 isTreeLoaded

This method verifies that the default view layers are already created and loaded into the view control (loadTree method was called).

Return value

Boolean

6.2.3.3 getParentNode

Returns the SVG Node related to the control.

Return value

DOM Node

6.2.3.4 setGroupsLayerVisibility

Allows the Groups layer to be either shown or hidden.

Input parameters

Name	Type	Description
isVisible	Boolean	Required layer visibility state.

6.2.3.5 setTreeLayerData

Sets new data for the layer.

Input parameters

Name	Type	Description
layerId	String | Number	Layer id or index, which should be updated.
newData	Any	New layer data.

6.2.3.6 getLeafDataNodesCount

Gets count of leaf data items from TreeLayer.

Return value

Number of leafs.

6.2.3.7 setScale

Gets count of leaf data items from TreeLayer.

Input parameters

Name	Type	Description
newScale	Number	Sets new scale value for layered view control.

6.2.3.8 getScale

Get count of leaf data items from TreeLayer.

Return value

Number

6.2.3.9 setSpacing

Sets new spacing value for Tree layer. This parameter effects distance between rendered item nodes, which are on adjacent hierarchy levels (horizontal distance).

Input parameters

Name	Type	Description
spacingValue	Number	Spacing distance in pixels

6.2.3.10 centerView

Calculates and sets the optimal view scale, when layers content fits client viewport.

6.2.3.11 adjustViewBounds

Recalculates SVG node size based on current view content bounds and scale. This is a complex operation, which should be used after scaling or significant content changes.

6.2.4 Events

The following sections describe the events that can be used as part of the VariantsTree Control.

6.2.4.1 onScaleChanged

Event fires when view scale is changed.

Input parameters

Name	Type	Description
newScale	Number	Scale value that was applied.

6.2.4.2 onGroupsSwitched

Event fires when view scale is changed.

Input parameters

Name	Type	Description
newScale	Number	Scale value that was applied.

6.2.5 Context Menu Events

The events described in this section belong to the dojo context menu widget. Use the dojo.connect and dojo.on modules to attach handlers for them.

6.2.5.1 onTreeMenuInit

Event handler for onMenuInit, which generates array of descriptors for all available context menu actions.

Return value

Array of menu items descriptors.

6.2.5.2 onTreeMenuSetup

Event handler for setup context menu item's state based on target layer and node.

Input parameters

Name	Type	Description
menuControl	Object, MenuWidget	Menu control, which should be updated.
targetLayer	Object, VisualizationLayer	Layer that contains Node for which menu should be displayed.
targetDataNode	Object	Data object that bound to the target DOM Node.

6.2.5.3 onMenuItemClick

Handler for onMenuItemClick event.

Input parameters

Name	Type	Description
commandId	String	Menu item id.
targetDataNode	Object	Data object that is bounded to the target DOM Node.