API Usage

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

- Configurator Services API methods
- AML Request Syntax
- AML Response Formats
- AML Request/Response examples

Along with AML, Configurator Services API methods can be called using Aras Innovator's RESTful API.

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:

- **cfg_GetScopeStucture**. This method creates a representation of the Scope object: it uses scope_builder to build the Scope object, which is then serialized into either AML or JSON. The method can be used to:
  - display a list of available choices
  - 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. Valid Combination is a set of variables with assigned values to them, which leads to valid solution. 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

API Methods

cfg_GetScopeStructure

This method retrieves a Scope Structure. To request the Scope Structure, invoke the method with an AML or OData request. The method returns the Scope Structure in AML or JSON respectively.

The output allows customizations to include extended information about Items.

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_builder_method>%output_builder_method%</output_builder_method>
</Item>
```

Where:

<targetScope /> %*builder method name*%	Refer to Input Item Format. The <targetScope /> node is required. For Missing Terms affect, refer to Influence of Missing Terms on the API results.
%*builder method attributes*%	Builder method attributes.
%*builder method properties*%	Builder method properties.
%*output_builder_method*%	Optional parameter to provide the Name of a server method that will override GetScopeStructureOutputBase using the Aras.Server.Core.Configurator.ScopeStructureOutput template. The server method is used to extend responses using custom data. The AML targetScope item is accessible in the output_builder_method.

AML Response 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>
```

OData Request Syntax

```
{
  "targetScope": {
    "Item": {
      "@aras.type": "Method",
      "@aras.action": "%builder method name%",
      "@aras.%builderMethodAttributeName1%": "%attribute 1 value%",
      "@aras.%builderMethodAttributeName2%": "%attribute 2 value%",
      ...
      "%builderMethodPropertyName1%": "%property 1 value%",
      "%builderMethodPropertyName2%": "%property 2 value%",
      ...
    }
  },
  "output_builder_method": "%output_builder_method%"
}
```

Important

For more information about using OData protocol, refer to the *Aras Innovator – RESTful API*.

Where:

targetScope %builder method name%	Refer to Input Item Format. The “ `targetScope` ” object is required.
%builderMethodPropertyName#%	A builder method property.
%builderMethodAttributeName#%	A builder method attribute.
%output_builder_method%	An Optional parameter to provide the Name of a server method that will override GetScopeStructureOutputBase using the Aras.Server.Core.Configurator.ScopeStructureOutput template. The server method is used to extend responses using custom data. The AML targetScope item is accessible in the output_builder_method.

JSON Response Format

```
{
  "Item": {
    "@aras.type": "Scope",
    "@aras.id": "ScopeId",
    "name": "ScopeName",
    "Relationships": {
      "Item": [
        {
          "@aras.type": "Variable",
          "@aras.id": "VariableId_1",
          "id": "VariableId_1",
          "name": "VariableName_1",
          "Relationships": {
            "Item": [
              {
                "@aras.type": "NamedConstant",
                "@aras.id": "NamedConstantId_1",
                "id": "NamedConstantId_1",
                "name": "NamedConstantName_1"
              },
              {
                "@aras.type": "NamedConstant",
                "@aras.id": "NamedConstantId_2",
                "id": "NamedConstantId_2",
                "name": "NamedConstantName_2"
              }
            ]
          }
        },
        ...
        {
          "@aras.type": "Rule",
          "@aras.id": "RuleId_1",
          "id": "RuleId_1",
          "name": "RuleName_1",
          "definition": "expression"
        },
        ...
      ]
    }
  }
}
```

Examples

This section provides examples of AML and OData requests as well as AML and JSON responses for the same Scope Structure retrieved by the cfg_GetScopeStructure method.

- ***AML Request***

```
<Item type="Method" action="cfg_GetScopeStructure">
  <targetScope>
    <Item type="Method" id="item_id_componentA" action="builder_method_name
    </Item>
  </targetScope>
</Item>
```

- ***AML Response***

```
<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>
```

- ***OData Request***

```
{
  "targetScope": {
    "Item": {
      "@aras.type": "Method",
      "@aras.action": "builder_method_name",
      "@aras.id": "item_id_componentA"
    }
  }
}
```

- ***JSON Response***

```
{
  "Item": {
    "@aras.type": "Scope",
    "@aras.id": "item_id_componentA",
    "name": "component A",
    "Relationships": {
      "Item": [
        {
          "@aras.type": "Variable",
          "@aras.id": "item_id_color",
          "id": "item_id_color",
          "name": "Color",
          "Relationships": {
            "Item": [
              {
                "@aras.type": "NamedConstant",
                "@aras.id": "item_id_red",
                "id": "item_id_red",
                "name": "Red"
              },
              {
                "@aras.type": "NamedConstant",
                "@aras.id": "item_id_green",
                "id": "item_id_green",
                "name": "Green"
              }
            ]
          }
        },
        {
          "@aras.type": "Variable",
          "@aras.id": "item_id_wheelsize",
          "id": "item_id_wheelsize",
          "name": "Wheel Size",
          "Relationships": {
```


```
            "Item": [
              {
                "@aras.type": "NamedConstant",
                "@aras.id": "item_id_15inch",
                "id": "item_id_15inch",
                "name": "15 inch"
              },
              {
                "@aras.type": "NamedConstant",
                "@aras.id": "item_id_16inch",
                "id": "item_id_16inch",
                "name": "16 inch"
              }
            ]
          }
        },
        {
          "@aras.type": "Rule",
          "@aras.id": "item_id_rule1",
          "id": "item_id_rule1",
          "name": "Rule Name 1",
          "definition": "expression"
        }
      ]
    }
  }
}
```

cfg_GetValidCombinations

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

Parameters

Required	Optional
targetScope	fetch, offset, select, responseFormat (default is JSON), condition For Missing Terms affect, refer to Influence of Missing Terms on the API results.

AML Request Syntax

The AML request syntax is as follows:

```
<Item type="Method" action="cfg_GetValidCombinations" select="%variableIds%" offset="%offset_integer%" fe
  <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 cfg_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. For Missing Terms affect, refer to
targetScope %*builder method name*% %*builder method attributes*% %*builder method properties*%	Refer to 4.3.2 Input Item Format targetScope is required. For Missing Terms affect, refer to .

AML Response Format

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 or an array of integers) property. The value of this property is used as an index or a set of indexes for the Values array.

Important

The BehaviorType of a Variable determines the type of its Combination value. For *exactly-one* BehaviorType, the Combination value is a single integer. For *at-least-one* BehaviorType, the Combination value is an array of integers. For *at-most-one* BehaviorType, the Combination value can be either an empty array or an array with a single integer.

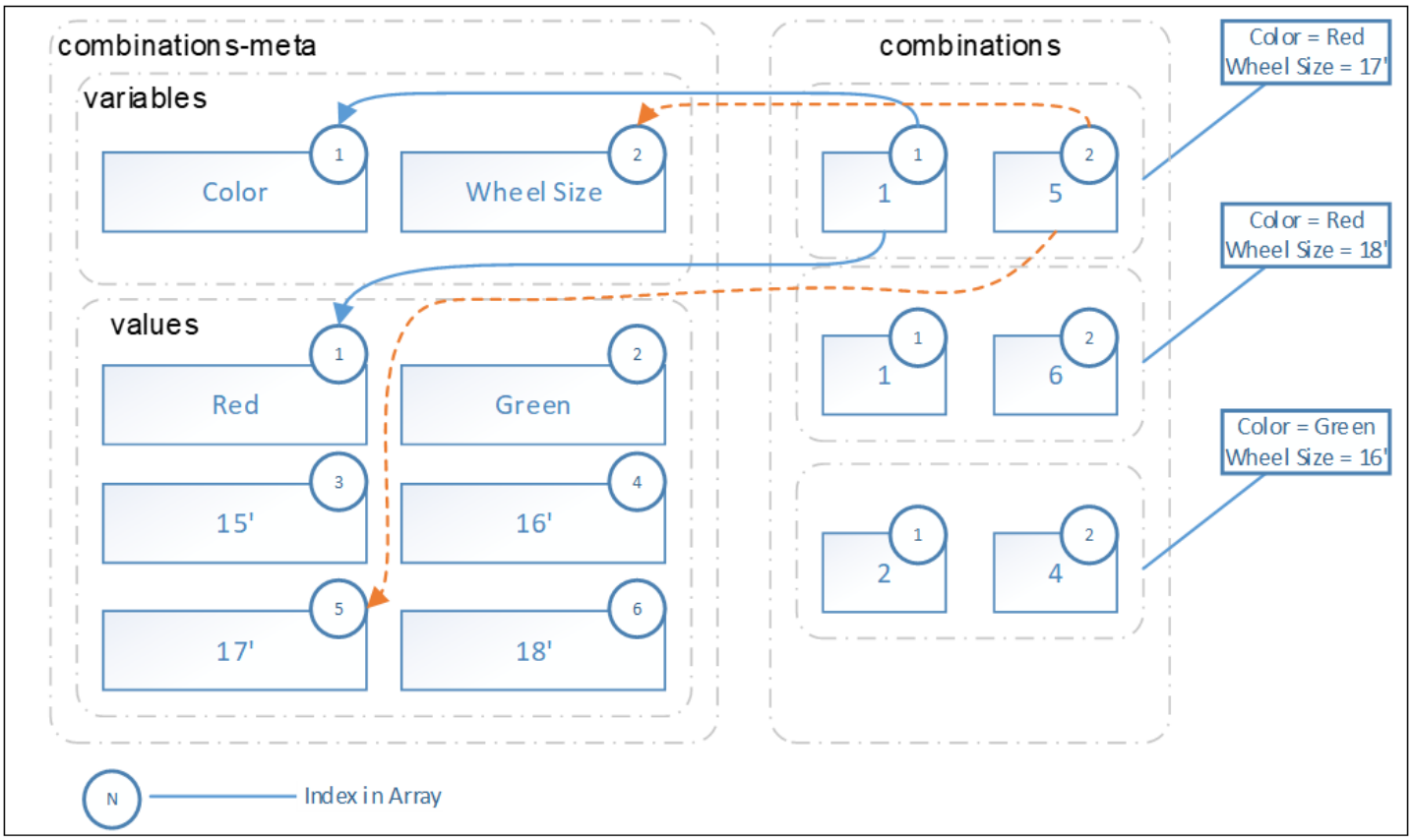
Important

The *exactly-one* BehaviorType is the default for a Variable. The response formats and examples are provided for this behavior type. Examples with responses for the *at-least-one* and *at-most-one* BehaviorTypes can be found in "Variable with BehaviorType equals at-least-one" and "Variable with BehaviorType equals at-most-one".

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]];
}
```



XML Format

The following response in XML Format is a response object 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>
    </result>
  </soap-env:Body>
</soap-env:Envelope>
```

JSON Format

The following response in JSON Format is a response object serialized to JSON 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": {
            "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>
```

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">
    </Item>
  </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_17inch</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>2</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_17inch"
            }
          ]
        },
        "combinations": [
          [0, 2],
          [0, 3],
          [1, 3],
          [1, 2]
        ]
      }
    </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" responseFor
  <targetScope>
    <Item type="Method" action="builder_method_name" id="business_item_id">
    </Item>
  </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
  <targetScope>
    <Item type="Method" action="builder_method_name" id="business_item_id">
    </Item>
  </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>
```

- ***Missing Terms in condition***
  The request contains "item_id_yellow" named constant, which is not included in the target Scope:

```
<Item type="Method" action="cfg_GetValidCombinations" select="item_id_color
  <condition><![CDATA[<expression>
    <eq>
      <variable id="item_id_color" />
      <named-constant id="item_id_yellow" />
    </eq>
  </expression>]]></condition>
  <targetScope>
    <Item type="Method" action="builder_method_name" id="business_item_id">
    </Item>
  </targetScope>
</Item>
```

The XML response:

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

Refer to “5.3 Influence of Missing Terms on the API results”.

- ***Variable with BehaviorType equals at-least-one***
  The target Scope contains only one variable with BehaviorType=AtLeastOne with two named-constants.
  The request is:

```
<Item type="Method" action="cfg_GetValidCombinations" responseFormat="XML">
  <targetScope>
    <Item type="Method" action="builder_method_name" id="business_item_id">
    </Item>
  </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_comfort" order="0" />
        </variables>
        <values>
          <value type="NamedConstant" order="0">item_id_heatedseats</value>
          <value type="NamedConstant" order="1">item_id_cooledseats</value>
        </values>
      </combinations-meta>
      <combinations>
        <combination>
          <value>0</value>
        </combination>
        <combination>
          <value>1,0</value>
        </combination>
        <combination>
          <value>1</value>
        </combination>
      </combinations>
    </result>
  </soap-env:Body>
</soap-env:Envelope>
```

The JSON response:

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Body>
    <result>
      {
        "combinations-meta": {
          "variables": {
            "item_id_comfort": 0
          },
          "values": [
            {
              "type": "NamedConstant",
              "id": "item_id_heatedseats"
            },
            {
              "type": "NamedConstant",
              "id": "item_id_cooledseats"
            }
          ]
        },
        "combinations": [
          [[0]],
          [[1, 0]],
          [[1]]
        ]
      }
    </result>
  </soap-env:Body>
</soap-env:Envelope>
```

- ***Variable with BehaviorType equals at-most-one***

  The target Scope contains only one variable with BehaviorType=AtMostOne with two named-constants.

  The request is:

  ```
  <Item type="Method" action="cfg_GetValidCombinations" responseFormat="XML">
    <targetScope>
      <Item type="Method" action="builder_method_name" id="business_item_id">
      </Item>
    </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_comfort" order="0" />
        </variables>
        <values>
          <value type="NamedConstant" order="0">item_id_heatedseats</value>
          <value type="NamedConstant" order="1">item_id_cooledseats</value>
        </values>
      </combinations-meta>
      <combinations>
        <combination>
          <value></value>
        </combination>
        <combination>
          <value>0</value>
        </combination>
        <combination>
          <value>1</value>
        </combination>
      </combinations>
    </result>
  </soap-env:Body>
</soap-env:Envelope>
```

The JSON response:

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope
  <soap-env:Body>
    <result>
      {
        "combinations-meta": {
          "variables": {
            "item_id_comfort": 0
          },
          "values": [
            {
              "type": "NamedConstant",
              "id": "item_id_heatedseats"
            },
            {
              "type": "NamedConstant",
              "id": "item_id_cooledseats"
            }
          ]
        },
        "combinations": [
          [[]],
          [[0]],
          [[1]]
        ]
      }
    </result>
  </soap-env:Body>
</soap-env:Envelope>
```

cfg_ValidateScope

The cfg_ValidateScope 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.

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 *Input Item Format* targetScope is required. For Missing Terms affect, refer to *Influence of Missing Terms on the API results*

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>
```

2. 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>
```

3. 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>
```

cfg_GetIntersectingExpressions

The cfg_GetIntersectingExpressions 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.

Important

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

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>
```

Important

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***	Refer to *Input Item Format*. targetScope is required. For information about Missing Terms, refer to *Influence of Missing Terms on the API results*.

attributes% %builder method properties%	
<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. For Missing Terms affect, refer to *Influence of Missing Terms on the API results*.

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 intersecting if there is at least one valid combination for Scope. The following examples demonstrate how the algorithm prepares sets of expressions:

In this example, there are two sets of expressions, and each set contains one element:

A (a1), B (b1)

The algorithm creates the following set:

(a1, b1), (b1, a1)

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

(a1, b1)

In the following example, there are five 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 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 three 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 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)

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>
```

Important

A minimum of two expressions is required 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 *Input Item Format* targetScope is required For Missing Terms affect, refer to *Influence of Missing Terms on the API results*
<cartesian-square>…</cartesian-square>	Node that contains 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.

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.

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>
    </result>
  </soap-env:Body>
</soap-env:Envelope>
```

JSON format

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

Examples

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

```
<Item type="Method" action="cfg_GetIntersectingExpressions" responseFormat=
  <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=
  <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>
```

Let us say the cfg_GetIntersectingExpressions method confirms that a Scope has the following available combinations:

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=
  <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 us say the cfg_GetIntersectingExpressions method confirms that a Scope has the following available combinations:

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>
```

cfg_GetConflicts

The API method described in this section will help you to find the reason why the Scope does not have a solution. Use the cfg_GetConflicts 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).

AML Request Syntax

The AML request syntax is as follows:

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

%fetch_integer%	Defines the number of conflicts that are returned.The Default value is empty, which returns all conflicts.It is suggested to use a low number for the 'fetch' based on how many conflicts need to be returned based on the business use cases. If a high or no value is specified for 'fetch' and the scope is complex, the execution time to return these conflicts will be high.
targetScope **- *%builder method name%*** **- *%builder method attributes%*** **- *%builder method properties%***	Refer to Input Item Format. targetScope is required. For Missing Terms affect, refer to Influence of Missing Terms on the API results.
<condition>…</condition>	A 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. For Missing Terms affect, refer to Influence of Missing Terms on the API results.

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[...]]></member>
        <member source="variable" id="..." propositionalform="..."><![CDATA[...]]></member>
        <member source="rule" id="..." propositionalform="..."><![CDATA[...]]></member>
        <member source="missinginscope" propositionalform="..."><![CDATA[...]]></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.

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>
    </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: Since 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 THE
      </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 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=G
          <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
      </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 cannot be any of these 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
          <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
      </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
          <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 THE
      </Conflict>
      <Conflict>
        <member source="condition" propositionalForm='Style=Wagon AND [Whee
          <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 T
      </Conflict>
      <Conflict>
        <member source="condition" propositionalForm='[Wheel Type]=Steel AN
```

```
          <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]=
      </Conflict>
    </Result>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Using the @fetch attribute to get one conflict
  Request:

```
<Item type="Method" action="cfg_GetConflicts" fetch="1">
        <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:

```
<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
          <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 THE
      </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=Sma
          <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-
      </Conflict>
      <Conflict>
        <member source="condition" propositionalForm="Size=Big AND Size=Sma
          <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="IDOFCOLOR" propositionalForm="EXACTLY
        <member source="rule" id="XXYYZZ" propositionalForm="IF (Size=Big)
        <member source="rule" id="UUVVWW" propositionalForm="IF (Size=Small
      </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=W
          <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-
        <member source="variable" id="IDOFCOLOR" propositionalForm="EXACTLY
        <member source="rule" id="XXYYZZ" propositionalForm="IF (Size=Big)
      </Conflict>
      <Conflict>
```

```
    <member source="condition" propositionalForm="(Size=Big AND Color=W
      <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-
    <member source="variable" id="IDOFCOLOR" propositionalForm="EXACTLY
    <member source="rule" id="UUVVWW" propositionalForm="IF (Size=Small
  </Conflict>
  <Conflict>
    <member source="condition" propositionalForm="(Size=Big AND Color=W
      <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-
      <member source="rule" id="XXYYZZ" propositionalForm="IF (Size=Big)
      <member source="rule" id="UUVVWW" propositionalForm="IF (Size=Small
    </Conflict>
    <Conflict>
      <member source="condition" propositionalForm="(Size=Big AND Color=W
        <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
        <member source="rule" id="XXYYZZ" propositionalForm="IF (Size=Big)
        <member source="rule" id="UUVVWW" propositionalForm="IF (Size=Small
      </Conflict>
    </Result>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Missing terms
  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>
```

This term causes the conflict, the corresponding `<member>` nodes to reflect this fact (refer to Influence of Missing Terms on the API results):

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope
  <SOAP-ENV:Body>
    <Result>
      <Conflict>
        <member source="condition" propositionalForm="Color=Yellow"><![CDAT
          <expression>
            <EQ>
              <variable id="IDOFCOLOR" />
              <named-constant id="YELLOW" />
            </EQ>
          </expression>
        ]]></member>
        <member source="missinginscope" propositionalForm="NOT(Color=Yellow
      </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>
  </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
        <member source="rule" id="UUVVWW" propositionalForm="IF (Color=Blac
      </Conflict>
    </Result>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Input condition expressions are not necessary to cause conflicts.

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.

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.

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 cfg_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>
```

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[
    <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>
  ]]></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>
```

Extra - How To

Get Unreachable Combinations

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 cfg_GetValidCombinations method.

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>
```

Unreachable combinations condition node should look like:

```
<!-- The original <expression>...</expression> content is moved inside <
     then the whole thing is wrapped in a CDATA-escaped <condition> elem
<condition>
  <![CDATA[
    <expression>
      <NOT>
        ...
      </NOT>
    </expression>
  ]]>
</condition>
```

Several separate expressions

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

Unreachable combinations condition node should look like:

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

Response Explanation

The cfg_GetValidCombinations method returns combinations that are not covered by any of the initial expressions. If the cfg_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.

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, 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 are not 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, 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>
    </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>
```

Get Unreachable Expressions

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 cfg_GetValidCombinations method.

Prepare Expression

There is no specific format for the ‘condition’ node of the cfg_GetValidCombinations method request. You must put your expression inside the ‘condition’ node. The following diagram shows how to create a proper ‘condition’ node.

Unreachable expression condition node should look like:

An expression

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

```
<condition>
  <![CDATA[
    <expression>
      ...
    </expression>
  ]]>
</condition>
```

Response Explanation

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

means the specified expression is unreachable.

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, 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" responseFor
   <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 '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>
```

Important

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_GetValidCombinations.

Influence of Missing Terms on the API results

Some variables or their constants may not be found within the target scope during transformations. In these cases, the term Missing Term is used. Missing Term is a Term which contains at least one ID that is not in the target Scope. Any Missing Terms transform into a “False” constant value while solving a Scope.

- ***Example***
  The target Scope contains the “Color” variable, which contains “Red” and “Black” Named Constants:

```
<Item type="Scope" id="ScopeId">
  <name>ScopeName</name>
  <relationships>
    <Item type="Variable" id="ColorVariableId">
      <id>ColorVariableId</id>
      <name>Color</name>
      <relationships>
        <Item type="NamedConstant" id="NamedConstantId_1">
          <id>NamedConstantId_1</id>
          <name>Red</name>
        </Item>
        <Item type="NamedConstant" id="NamedConstantId_2">
          <id>NamedConstantId_2</id>
          <name>Black</name>
        </Item>
      </relationships>
    </Item>
  </relationships>
</Item>
```

  Consider the following expression concerning the target Scope:

```
<expression>
  <eq>
    <variable id="ColorVariableId" />
    <named-constant id="NamedConstantId_1" />
  </eq>
  <eq>
    <variable id="ColorVariableId" />
    <named-constant id="NamedConstantId_2" />
  </eq>
  <eq>
    <variable id="ColorVariableId" />
    <named-constant id="NamedConstantId_3" />
  </eq>
</expression>
```

In this case, the following term from the expression is considered as a Missing Term:

```
<eq><variable id="ColorVariableId" /><named-constant id="NamedConstantId_3'
```

As result, the whole expression will be converted to “False”.