Aras Innovator 11

Effectivity Services Programmer’s Guide

Document #: 11.0.02018030901

Last Modified: *9/12/2018*

Copyright Information

Copyright © 2018 Aras Corporation. All Rights Reserved.

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/

Notice of Rights

Copyright © 2018 by Aras Corporation. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, V1.0 or later (the latest version is presently available at http://www.opencontent.org/openpub/).

Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder.

Distribution of the work or derivative of the work in any standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from the copyright holder.

Aras Innovator, Aras, and the Aras Corp "A" logo are registered trademarks of Aras Corporation in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

Notice of Liability

The information contained in this document is distributed on an "As Is" basis, without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement. Aras shall have no liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this document or by the software or hardware products described herein.

Table of Contents

Send Us Your Comments 6

1 Introduction 8

1.1 What is Effectivity Services? 8

2 Terminology 9

3 Getting Started 10

3.1 Effectivity Services Data Model Concept Overview 10
3.2 Effectivity Services Process Flow Overview 11
3.3 Process Flow Overview of Effectivity Services combined with Query Builder and Tree Grid View 13

4 Free-Form Boolean Expression Language 15

4.1 Expression Nodes 16

4.1.1 <expression>…</expression> 16
4.1.2 <eq>…</eq> 17
4.1.3 <ge>…</ge> 17
4.1.4 <le>…</le> 17
4.1.5 <variable id="…"/> 18
4.1.6 <named-constant id="…"/> 18
4.1.7 <constant>…</constant> 18
4.1.8 <and>…</and> 18
4.1.9 <or>…</or> 19
4.1.10 <not>…</not> 20
4.1.11 <implication>…</implication> 20
4.1.12 <exactly-one/> 21
4.1.13 <at-most-one /> 22
4.1.14 <at-least-one /> 22

5 Data Model 23

6 Configuring Effectivity Services 26

6.1 Administrator Actions 26
6.2 Administrator/User Actions 27
6.3 User Actions 27

7 Scope Object Model 28

7.1 Scope Object Model Design 28
7.2 Customizing Scope Builder 29

7.2.1 Implementing the Scope Builder Method 29
7.2.2 Builder Method usage 29
7.2.3 Scope Resolver 29
7.2.4 Caching 30
7.2.5 Sample Builder Method 31

©2018 Aras Corporation All Copyrights Reserved.

8 Admin Setup 33

9 API Usage 37

9.1 Overview 37
9.2 Creating a Query Definition to Filter by Effectivity 38

9.2.1 Setting up a Query Definition 39
9.2.2 Defining Effectivity Criteria 40

9.3 Creating a Tree Grid View to Display Effective Items 45

9.3.1 Setting Up a Tree Grid View 45
9.3.2 Using Tree Grid View Parameters for Resolving Effective Items 53

10 Dynamic Tree Grid Control API 71

10.1 Constructor 71
10.2 Public fields 71
10.3 Events 71

10.3.1 addRow 72
10.3.2 addRows 72
10.3.3 removeRow 72
10.3.4 removeAllRows 73
10.3.5 addColumn 73
10.3.6 removeColumn 73

10.4 Common objects description 74

10.4.1 Metadata objects 74
10.4.2 Column settings object 74
10.4.3 Column object 75
10.4.4 Row object 75

10.5 Public methods 76

10.5.1 loadData 76
10.5.2 obtainRowID 77
10.5.3 addRows 78
10.5.4 removeRow 78
10.5.5 removeAllRows 79
10.5.6 selectRow 79
10.5.7 deselectRow 79
10.5.8 addColumn 79
10.5.9 removeColumn 80
10.5.10 getChildRowIds 80
10.5.11 getRowUserData 80
10.5.12 setRowUserData 81
10.5.13 setCellEditability 81
10.5.14 getCellEditability 82
10.5.15 setCellValue 82
10.5.16 getCellValue 82
10.5.17 getColumnCount 83
10.5.18 getVisibleColumnCount 83
10.5.19 getColumnIndex 83
10.5.20 getColumnName 83

©2018 Aras Corporation All Copyrights Reserved.

10.5.21 getColumnOrder ... 83
10.5.22 getAllColumnNames ... 84
10.5.23 getParentId ... 84
10.5.24 containsRow ... 84
10.5.25 containsColumn ... 84
10.5.26 expandAll ... 85
10.5.27 collapseAll ... 85
10.5.28 isRowExpanded ... 85
10.5.29 isRowVisible ... 85
10.5.30 getVisibleRowCount ... 85
10.5.31 getAllRowCount ... 85
10.5.32 getVisibleRowIds ... 86
10.5.33 getAllRowIds ... 86
10.5.34 getRowId ... 86
10.5.35 getRowIndex ... 86
10.5.36 moveColumn ... 87
10.5.37 setColumnSettings ... 87
10.5.38 getColumnSettings ... 87
10.5.39 setColumnMetadata ... 87
10.5.40 getColumnMetadata ... 88
10.5.41 setColumnVisibility ... 88
10.5.42 getColumnVisibility ... 88
10.5.43 setColumnLabel ... 90
10.5.44 getColumnLabel ... 90
10.5.45 setColumnWidth ... 90
10.5.46 getColumnWidth ... 91
10.5.47 setCellMetadata ... 91
10.5.48 getCellMetadataOnly ... 91

©2018 Aras Corporation All Copyrights Reserved.

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/

©2018 Aras Corporation All Copyrights Reserved.

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.
Successive menu choices	Successive menu choices may appear with a greater than sign (-->) between the items that you will select consecutively. Example: Navigate to **File** --> **Save** --> **OK**.

©2018 Aras Corporation All Copyrights Reserved.

aras

1 Introduction

"Effectivity" refers to the ability to conditionally include items in a structure. This condition can consist of multiple variables influencing the decision to include or exclude an item in a specific configuration of the structure.

Let's use building an assembly as an example. This assembly is included in a product structure. It uses Part-1 for product model "X" and Part-2 for product model "Y." In this case, product model is the variable used in defining the part's effectivity in this structure.

There are many different types of effectivity variables depending on the types of products, for example:

- The Aerospace and Shipbuilding industries create low-volume, highly complex products. Differences in product configurations are tracked by variables, such as Model, Unit Number, or Hull Number.
- The Automotive and High-Tech industries create high-volume products. Configuration changes are influenced by when materials are available to use. Therefore, Date-based effectivity is commonly used.
- The Food and Beverage and Pharmaceutical industries create formula-based products. Changes to the product configuration are tracked using Lot, Batch, or Plant effectivity.

1.1 What is Effectivity Services?

Effectivity Services is built into the Aras Platform. It provides core functionality to develop applications to manage effectivity on structures and resolve structures for given effectivity criteria.

Together, Effectivity core services and the custom application layer enable you to:

- Define custom effectivity variables (such as date, model, unit, lot, batch, and plant),
- Set effectivity conditions on relationships,
- Recursively resolve structures by effectivity to generate configured structures.

©2018 Aras Corporation All Copyrights Reserved.

2 Terminology

Term	Definition
Effectivity	Identification of valid uses of an item in a structure, if it is conditional.
Effectivity Variable	Variables that influence effectivity decisions, such as date, model, unit, batch, lot, plant.
Effectivity Scope	Built-in ItemType that represents a list of relevant Effectivity Variables. For example, configuration differences in standard bicycles may be tracked by Model and Date, while configuration differences in highly customized performance bicycles may be tracked by Unit Number.
Effectivity Expression	Expression representing the effectivity condition in the Boolean Expression Language. For example: Model = "Model X" and (Unit >= 10 and Unit <=20)
Effectivity Criteria	Criteria used for resolving a structure by effectivity. Effectivity conditions set on relationships are evaluated against the input criteria to determine the inclusion or exclusion of conditional items in the resolved structure. For example: Model = "Model X", Unit =15
Scope Object	Instance of the class "Aras.Server.Core.Configurator.Scope", and is a container of Variable instances.

©2018 Aras Corporation All Copyrights Reserved.

3 Getting Started

Effectivity Services is used for resolving structures with effectivity conditions on Relationship Items for specified criteria.

The Aras Platform uses the following Effectivity Services features:

- The Effectivity Services Data Model built with new ItemTypes, RelationshipTypes, Methods and more to enable customization and/or extension to support a custom flow.
- The Effectivity Resolution Engine determines which items are effective in a structure for a specific configuration.

3.1 Effectivity Services Data Model Concept Overview

The Effectivity Services data model concept contains the following main structural blocks:

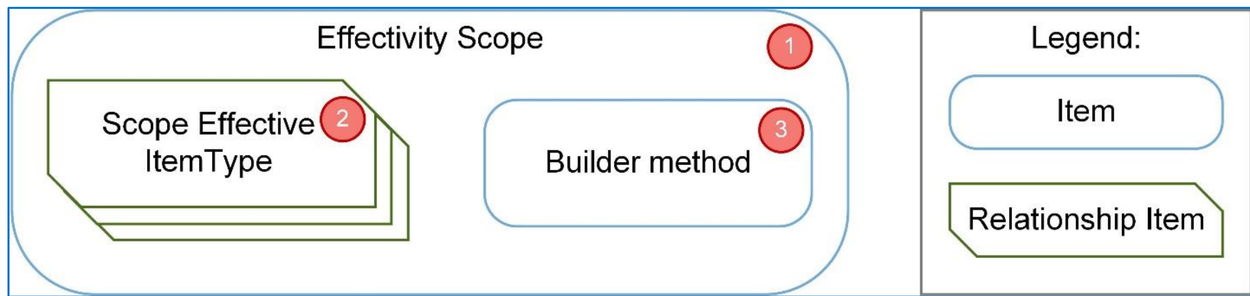


Figure 1.

1. **Effectivity Scope** is an item of the ItemType "effs_scope" used for defining the context for effectivity resolution. An Effectivity Scope item requires a reference to the "Builder method" and may have "Scope Effective ItemType" relations to the RelationshipTypes where effectivity is managed.
2. **Scope Effective ItemType** is an item of the RelationshipType "effs_scope_itemtype" which creates a relationship between an Effectivity Scope item and an ItemType ("RelationshipType" ItemType). When this relationship is established, the Effectivity Services Core automatically creates a new null (No Related) RelationshipType for the specified ItemType and names it according to the template "effs_%ITEMTYPE%_expression", where %ITEMTYPE% is the ItemType name.

Note: If the automatically generated RelationshipType name ("effs_%ITEMTYPE%_expression") is longer than 30 characters, you will be unable to create the relationship in the relationship grid. In this case, the Administrator must either manually enter the name on the **Effectivity Scope ItemType** form or send an AML request with the explicitly specified property "name" to create the RelationshipType. The length of the property "name" value should not exceed 13 characters.

The RelationshipType "effs_%ITEMTYPE%_expression" is added as a poly source to the polymorphic ItemType "effs_expression" in order to subscribe to the Effectivity Resolution Engine. This RelationshipType retains Effectivity Expressions.

©2018 Aras Corporation All Copyrights Reserved.

3. **Builder method** is an item of the ItemType "Method". A reference to it is stored in the property "builder_method" associated with an Effectivity Scope item. The Effectivity Resolution Engine calls this method to construct a Scope object, which serves as the base for effectivity resolution. For more information about the builder method, refer to section 7.2.1 Implementing the Scope Builder Method.

Note: The Effectivity Scope is an item of the ItemType "effs_scope". The Scope object is an instance of the class "Aras.Server.Core.Configurator.Scope," which is the result of Builder method execution.

3.2 Effectivity Services Process Flow Overview

The following diagram shows the process flow for Effectivity Services.

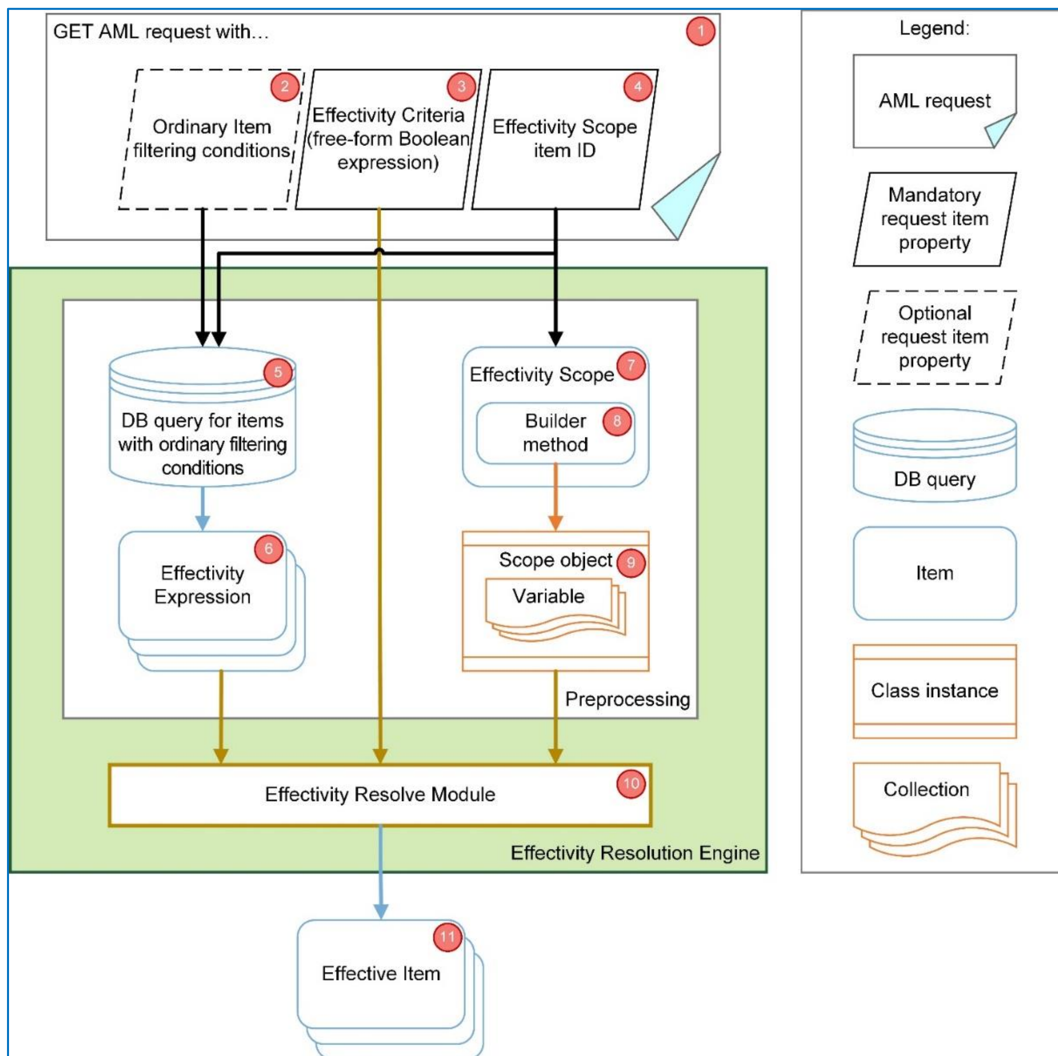


Figure 2.

©2018 Aras Corporation All Copyrights Reserved.

1. **An AML request with the action "get",** where there are mandatory and optional request item properties (nodes) to apply effectivity resolution on a requested structure.

   Note: The Effectivity Resolution Engine intercepts only "GET" AML requests.

   The current implementation of the Effectivity Resolution Engine can intercept "GET" AML requests to the ItemTypes, which are poly sources of the polymorphic ItemType "effs_expression". Therefore, it can process items of poly source ItemTypes of the polymorphic ItemType "effs_expression" to determine effective items.

2. **Ordinary item filtering conditions** in an AML request are used as criteria to limit the items requested from a database (DB) in the preprocessing stage of effectivity resolution processing. The GetItem internal module uses them to request items from the DB to fulfill specified filtering conditions. For example, these criteria may be item id, a list of item ids (known as the "idlist" attribute on a request Item node), or item property tags in an AML request. Ordinary item filtering conditions are optional and provided as needed.

3. **Effectivity Criteria** is a condition (e.g., Model = CH-X and Production Date = 2018/06/28) written in the Free-Form Boolean expression. For more information, refer to section 4 Free-Form Boolean Expression Language. The criteria are used to evaluate each effectivity expression in a requested structure to determine effective items. An item is effective within a structure if the evaluation of its corresponding Effectivity Expression(s) against Effectivity Criteria produces the logical truth.

   The representation of the "Effectivity Criteria" mandatory field in an AML request is the node "definition" matching the XPath "Item/definition | Item/and/definition" and containing a free-form Boolean expression in the XML-encoded form.

   Note: The node "definition" is not passed to the internal GetItem module. Therefore, it will not affect the items requested from a database.

4. **Effectivity Scope item ID** is the unique identifier (ID) of an Effectivity Scope Item used as a context for effectivity resolution.

   The representation of the "Effectivity Scope item ID" mandatory field in an AML request is the node "effs_scope_id" matching the XPath "Item/effs_scope_id | Item/and/effs_scope_id" and containing 32-hex GUID.

   Note: The node "effs_scope_id" is passed to the internal GetItem module as part of querying the database.

5. **DB query for items** with ordinary filtering conditions is an action on the preprocessing stage of effectivity resolution. The internal GetItem module requests items from a DB with optional item filtering conditions and the mandatory property "effs_scope_id". A response has items with effectivity expressions for evaluation with Effectivity Criteria.

6. **Effectivity Expression items** are the result of querying a DB for items, which fulfill specified optional item filtering conditions and the mandatory property "effs_scope_id".

7. **Effectivity Scope** is an item of the ItemType "effs_scope" requested from a DB using the incoming mandatory request property "effs_scope_id". This is to get a builder method referenced in the property "builder_method" of an Effectivity Scope item.

8. **Builder Method** is an item of the ItemType "Method". The preprocessing stage of Effectivity Resolution extracts the last generation of the method code using a reference in an Effectivity Scope item, compiles the code and calls it with the single argument "Effectivity Scope Id" to return a properly constructed Scope object.

©2018 Aras Corporation All Copyrights Reserved.

9. **Scope object** is a properly constructed instance of the class "Aras.Server.Core.Configurator.Scope", which has a list of Effectivity Variables, which will take a part in Effectivity Resolution.
10. **Effectivity Resolve Module** takes Effectivity Expression items, Effectivity Criteria, and a Scope object to evaluate each Effectivity Expression item against Effectivity Criteria. If this evaluation produces the logical truth, the item goes to the resulting structure.
11. **Effective items** are the result of the effectivity resolution process flow. The final structure only contains the items determined to be "effective".

3.3 Process Flow Overview of Effectivity Services combined with Query Builder and Tree Grid View

Combining Effectivity Services with the Query Builder and Tree Grid View applications is one of the native ways to visualize the results of the effectivity resolution process in the user interface.

The intended use of the Query Builder application is to create Query Definitions which are reusable AML requests. For more information about using Query Builder, refer to the *Aras Innovator 11.0 – Query Builder Guide*.

The Tree Grid View application enables you to build a visual data structure. For more information about Tree Grid View, refer to the *Aras Innovator 11.0 – Tree Grid View Administrator Guide*.

Effectivity Services acts as an interlayer in the native Query Definition process flow. The Effectivity Resolution Engine performs additional processing on the items requested by Query Definition.

The following diagram illustrates the process flow for Effectivity Services combined with Query Builder and Tree Grid View:

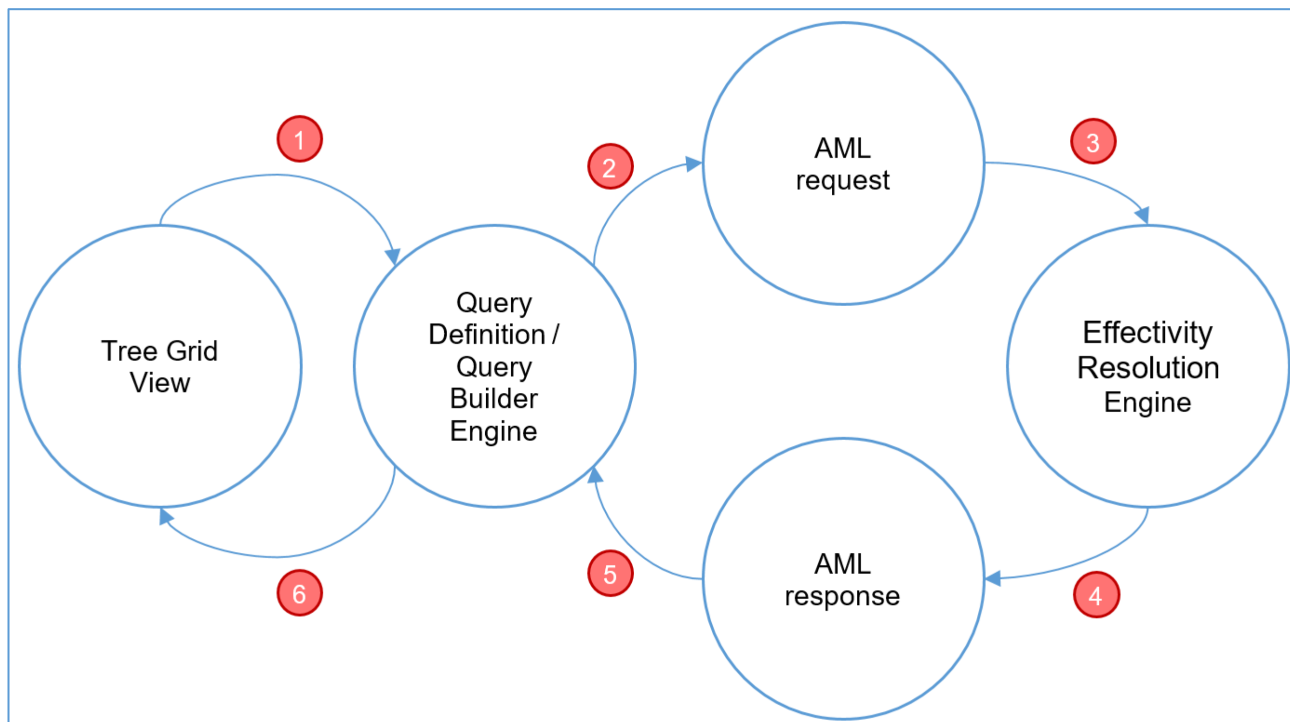


Figure 3.

©2018 Aras Corporation All Copyrights Reserved.

1. The configured **Tree Grid View** uses its associated Query Definition to query data to display. For information on how to configure the Tree Grid View, refer to section 9.3 Creating a Tree Grid View to Display Effective Items.
2. The **Query Builder Engine** processes the received Query Definition to retrieve data from a database (DB) and return it. To complete this task, the Query Builder Engine prepares and sends the AML request internally. For information on how to configure the Query Definition, refer to section 9.2 Creating a Query Definition.
3. The **Effectivity Resolution Engine** receives the AML request from the Query Builder Engine, gets the necessary data from a DB by applying the request, and filters data according to the Effectivity Criteria.
4. The **Effectivity Resolution Engine** prepares the AML response containing the effective items and sends it back to the Query Builder Engine.
5. The **Query Builder Engine** receives and processes the requested data according to the requirements of the regular query execution flow.
6. The **Query Builder Engine** sends the items structure to the Tree Grid View to display.

©2018 Aras Corporation All Copyrights Reserved.

4 Free-Form Boolean Expression Language

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

expression

eq

ge

le

variable

named-constant

constant

and

or

not

implication

condition

consequence

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

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

CDATA Example:

```
<containerNode>
    <![CDATA[<expression>
        <and>
          <eq>
            <variable id="item_id_model" />
            <named-constant id="item_id_z5" />
          </eq>
          <eq>
            <variable id="item_id_unit" />
            <constant type="int">10</constant>
          </eq>
        </and>
    </expression>]]>
</containerNode>
```

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;
```

©2018 Aras Corporation All Copyrights Reserved.

```
          &lt;variable id="item_id_model" /&gt;
          &lt;named-constant id="item_id_z5" /&gt;
        &lt;/eq&gt;
        &lt;eq&gt;
          &lt;variable id="item_id_unit" /&gt;
          &lt;constant type="int"&gt;10&lt;constant&gt;
        &lt;/eq&gt;
      &lt;/and&gt;
  &lt;/expression&gt;
</containerNode>
```

The following sections describe these nodes and include examples.

4.1 Expression Nodes

The following table lists dependencies between expression nodes.

Table 2:

Tag	Can contain
expression	implication, and, or, not, eq, ge, le
eq	variable, named-constant, constant
ge	variable, constant
le	variable, constant
variable	id attribute
named-constant	id attribute
constant	type
and	implication, and, or, not, eq, ge, le
or	implication, and, or, not, eq, ge, le
not	implication, and, or, not, eq, ge, le
implication	condition, consequence
condition	implication, and, or, not, eq, ge, le
consequence	implication, and, or, not, eq, ge, le

4.1.1 <expression>…</expression>

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

©2018 Aras Corporation All Copyrights Reserved.

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

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

4.1.2 <eq>…</eq>

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

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

This example represents the Boolean expression for Model = Z5.

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

This example represents the Boolean expression for Unit = 10.

4.1.3 <ge>…</ge>

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

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

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

4.1.4 <le>…</le>

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

©2018 Aras Corporation All Copyrights Reserved.

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

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

4.1.5 <variable id="…"/>

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

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

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

4.1.6 <named-constant id="…"/>

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

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

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

4.1.7 <constant>…</constant>

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

- Int
- DateTime
- String

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

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

4.1.8 <and>…</and>

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

©2018 Aras Corporation All Copyrights Reserved.

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

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

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

4.1.9 <or>…</or>

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

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

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

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

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

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

©2018 Aras Corporation All Copyrights Reserved.

4.1.10 <not>…</not>

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

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

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

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

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

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

4.1.11 <implication>…</implication>

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

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

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

©2018 Aras Corporation All Copyrights Reserved.

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

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

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

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

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

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

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

©2018 Aras Corporation All Copyrights Reserved.

Note: Each equivalence in EXACTLY-ONE must use the same variable in 11.0 SP14.

4.1.13 <at-most-one />

The `<at-most-one>` node defines an operation. The operator describes a condition that means "either none or just one of the listed equivalencies can be true". `<at-most-one>` must contain a set of `<eq>` child nodes. Each equivalence in AT-MOST-ONE must use the same variable.

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

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

Note: Each equivalence in AT-MOST-ONE must use same variable in 11.0 SP14.

4.1.14 <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>` must contain a set of `<eq>` child nodes: Each equivalence in AT-LEAST-ONE must use the same variable.

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

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

Note: Each equivalence in AT-LEAST-ONE must use the same variable in 11.0 SP14.

©2018 Aras Corporation All Copyrights Reserved.

5 Data Model

This section contains a technical description of the Effectivity Services Data Model.

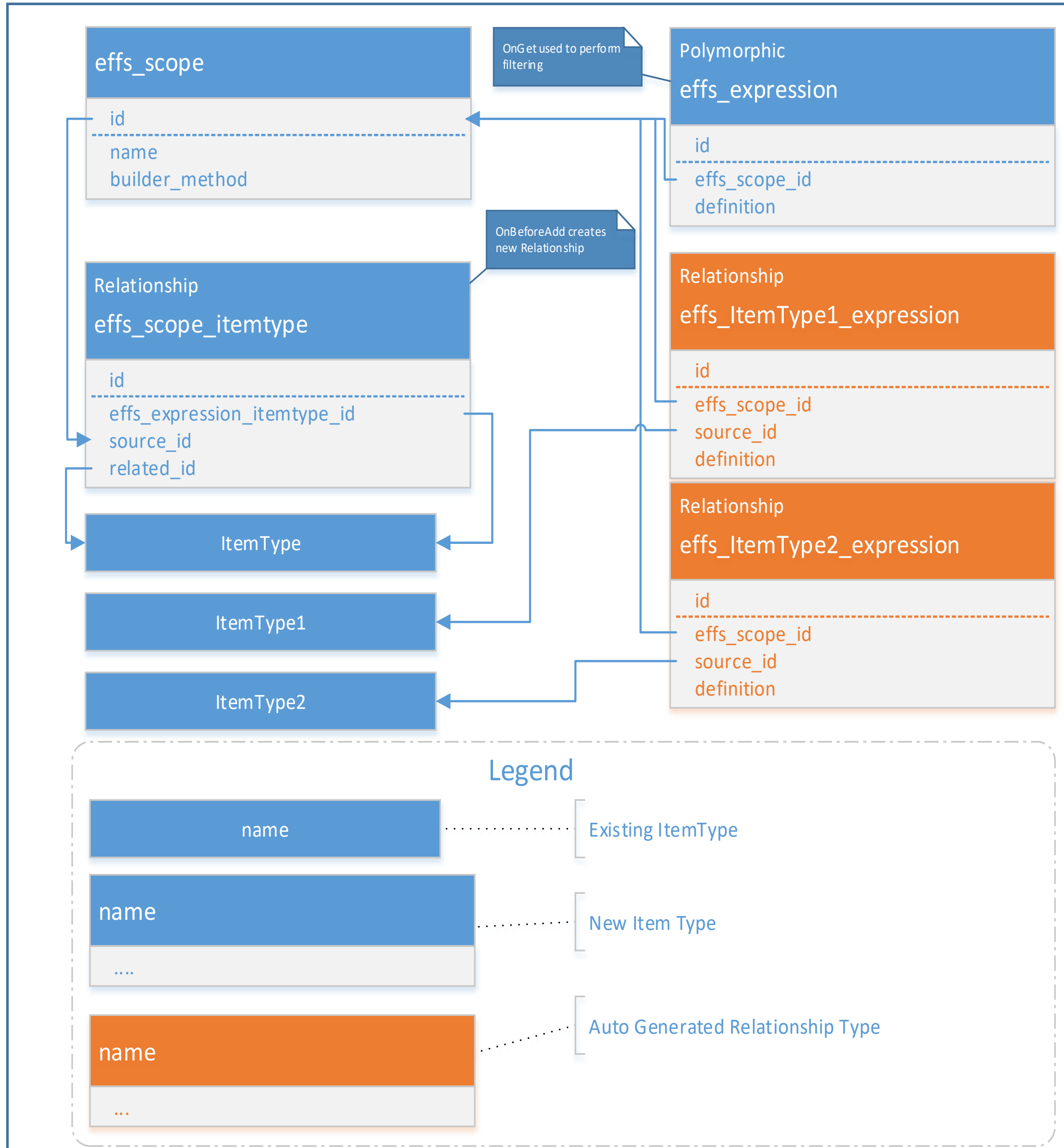


Figure 4.

©2018 Aras Corporation All Copyrights Reserved.

Table 3:

Item Type	Property	Description
ItemType		Standard ItemType ItemType, which is a holder of all ItemTypes in Aras Innovator
ItemType1, ItemType2		Relationship ItemTypes that are to be managed by effectivity.
effs_scope, effs_scope_itemtype, effs_expression		Effectivity Data Model
effs_scope		Container for scope object definition
	name	Name of Scope Item
	builder_method	Link to custom builder method that will construct Scope object for current Item
effs_scope_itemtype		Relationship between Scope (source_id) and target Item Type(related_id) onBeforeAdd event auto-generates no related relationship with name effs_#ItemType#_expression. Just one Item can be created for pair scope and item type.
	effs_expression_itemtype_id	itemType id, effs_#ItemType#_expression
	source_id	effs_scope id
	related_id	itemType id, target relationship Item Type
effs_expression		Polymorphic ItemType with onGet event
	effs_scope_id	effs_scope id
	definition	Container for expression

©2018 Aras Corporation All Copyrights Reserved.

aras

Item Type	Property	Description
effs_#ItemType#_expression		Poly source for effs_expression, No related relationship to target ItemType
	effs_scope_id	effs_scope id
	source_id	item id for target Item Type
	definition	Container for expression

©2018 Aras Corporation All Copyrights Reserved.

6 Configuring Effectivity Services

This section provides a brief description of what needs to be configured in Effectivity Services in order to be able to use it with Query Builder and Tree Grid View. A more detailed description can be found in subsequent sections.

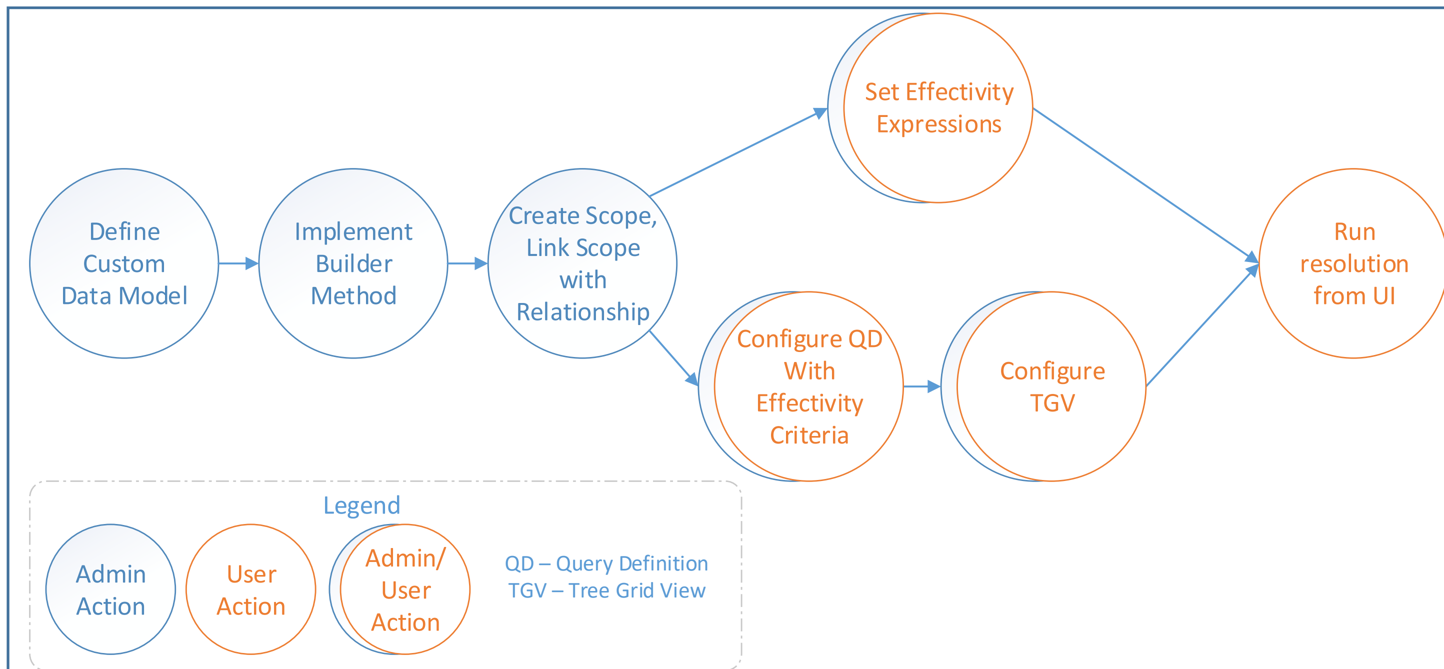


Figure 5.

6.1 Administrator Actions

Administrators are responsible for the following tasks:

- Define Custom Data Model – Define business item types to be used to build Effectivity Scope. These can be item types for effectivity variables, relationships where effectivity information is managed.
- Implement builder method - Implement server method that builds Scope object from custom Data.
- Create Scope – Add effs_scope Item, specify Name and builder method. Link custom Data to Scope if needed.
- Link Scope with Relationship – Add effs_scope_itemtype Item. Link Scope and Relationship ItemType that need to be managed by effectivity.

©2018 Aras Corporation All Copyrights Reserved.

6.2 Administrator/User Actions

Both Administrators and Users can perform the following actions:

- Set Effectivity Expressions – Create the effs_#ItemType#_expression Item with an Effectivity Expression for selected Relationship Item.
- Configure Query Definition to work with Effectivity Criteria – Create a Query Definition that selects your data using the specified criteria expression.
- Configure Tree Grid View to display returned data – Create a Tree Grid View configuration that shows the data returned from executing the Query Definition.

6.3 User Actions

Users can specify effectivity resolution criteria by entering parameter values for effectivity variables using the UI.

Note: Query Builder and Tree Grid View must be configured to work with specific parameters that enable entering values to create a structure resolution request.

©2018 Aras Corporation All Copyrights Reserved.

7 Scope Object Model

7.1 Scope Object Model Design

The Scope object contains a list of Variables. A Variable can be one of three data types (integer, date, string) or can have Enum with a list of values that can be assigned to the variable.

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

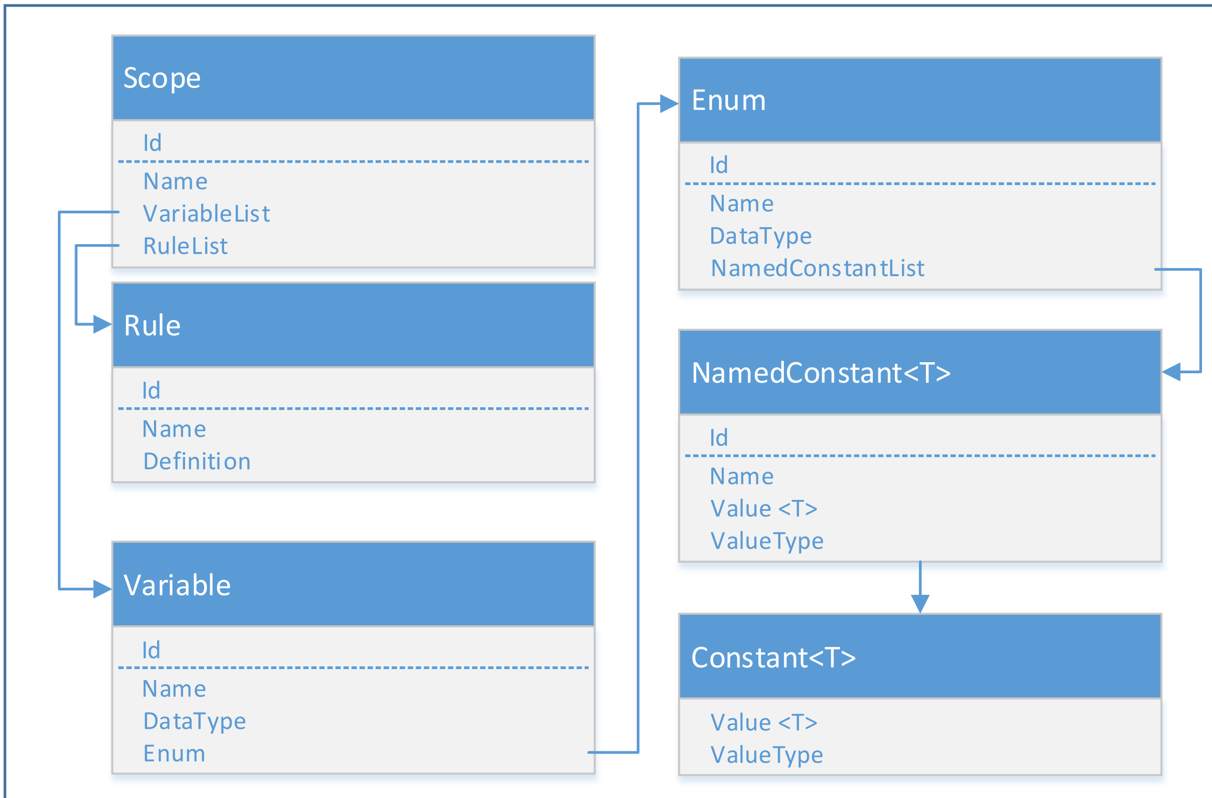


Figure 6.

Scope is a class that contains a list of Variables and a list of Rules.

Note: Effectivity Services does not support RuleList and Rule class in 11.0 SP14.

Variable classes store variable definitions. Variable datatypes are integer, date, string or null. The Variable type will be null if the Variable uses Enum with a list of Named constants. If Enum is null, the Variable is handled by type.

Enum is the container for the list of Named constants. A Named constant is intended to store the value definition. It contains the property name, value, and valuetype.

©2018 Aras Corporation All Copyrights Reserved.

7.2 Customizing Scope Builder

The Scope Builder method is a server method that is responsible for constructing a Scope Object. The purpose of the builder method is to construct a Scope Object using custom business data and business logic.

The Scope Builder uses a predefined method template. The template enables you to implement a Scope builder method properly.

7.2.1 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. Here 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: Implementing a Scope Builder method forces the override all abstract methods.

7.2.2 Builder Method usage

The Effectivity Scope ItemType (effs_scope) is associated with the property builder_method. This property is a link to the Method Item. Effectivity Services automatically runs the linked builder method to get the Scope Object for the specified Effectivity Scope Item.

7.2.3 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:

```
internal class ScopeResolverModule
    {
```

©2018 Aras Corporation All Copyrights Reserved.

```
        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) { }
    }
```

7.2.4 Caching

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

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

The Scope object can be invalidated automatically. Invalidation is triggered if at least one of the Items that the Scope object depends on changes. To do this, the system maintains a list of Item IDs and list of ItemTypes associated with these Items. Any time an item associated with a maintained ItemType and ID is changed, the system invalidates the found item.

The following collections must be created to make caching and invalidation possible:

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

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

  Each Item Type in this list must be a poly source item for the ScopeCacheDependency Poly Item ItemType.

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

  ```
  public override string[] GetGuidsItemDependsOn()
  {
  ```

©2018 Aras Corporation All Copyrights Reserved.

```
            return new string[] {
                    "item_id_Z4",
                    "item_id_Z5",
                    "item_id_Z6",
                    "item_id_Z7"
            };
        }
```

7.2.5 Sample Builder Method

The effs_scope ItemType has a property named builder_method. This Item property is the link to a custom server method. The Builder method is a server method that is responsible for constructing a Scope Object from custom business data.

This sample shows how to implement a Builder method that creates a static Scope Object.

```
//MethodTemplateName=CSharp:Aras.Server.Core.Configurator;
public override Scope BuildScope()
{
    Scope builtScope = new Scope { Id = "item_id_scope", Name = "Model/SN Scope" };
    Variable int_variable = new Variable(DataType.Int) { Id = "item_id_serialNumber", Name
= "Serial Number" };
    builtScope.VariableList.Add(int_variable);

    Aras.Server.Core.Configurator.Enum vEnum = new
Aras.Server.Core.Configurator.Enum(DataType.String) { Id = "item_id_modelList", Name =
"List of Models" };
    vEnum.AddNamedConstant("item_id_modelX", "Model X", "1");
    vEnum.AddNamedConstant("item_id_modelY", "Model Y", "2");
    vEnum.AddNamedConstant("item_id_modelZ", "Model Z", "3");
    builtScope.VariableList.Add(new Variable(DataType.String) { Id = "item_id_model", Name
= "Model", Enum = vEnum });

    return builtScope;
}

public override ArrayList GetCustomKey()
{
    return new ArrayList { "static_scope" };
}

public override string[] GetGuidsItemDependsOn()
{
    return new string[0];
}

public override List<string> GetItemTypeNamesItemDependsOn()
{
    return new List<string> { };
}
```

In this sample we have a static Scope Object, so we can store it in the cache using a hardcoded ID as a key. The method result is cached with key `"static_scope"`.

The Scope has two Variables:

- [Serial Number] – data type of integer

©2018 Aras Corporation All Copyrights Reserved.

- [Model] - data type of string, with list of values:
  - [Model X]
  - [Model Y]
  - [Model Z]

Note: All IDs in this sample code are hardcoded to show the meaning of the builder method. An actual implementation of a builder method retrieves business data and builds the appropriate Scope Object.

GetCustomKey, GetGuidsItemDependsOn, and GetItemTypeNamesItemDependsOn return either static or empty values for this sample static Scope Object. A Scope Object using real business data would return a feasible result.

©2018 Aras Corporation All Copyrights Reserved.

8 Admin Setup

This section describes how to configure Effectivity Services to manage Effectivity on selected Relationship ItemTypes.

1. You need to first create an Effectivity Scope item to define a context. Select **Administration>Effectivity Services>Effectivity Scope** in the TOC. You can search for existing Scope items by name or associated builder method. The Builder method contains a list of Variables that are available in the context.

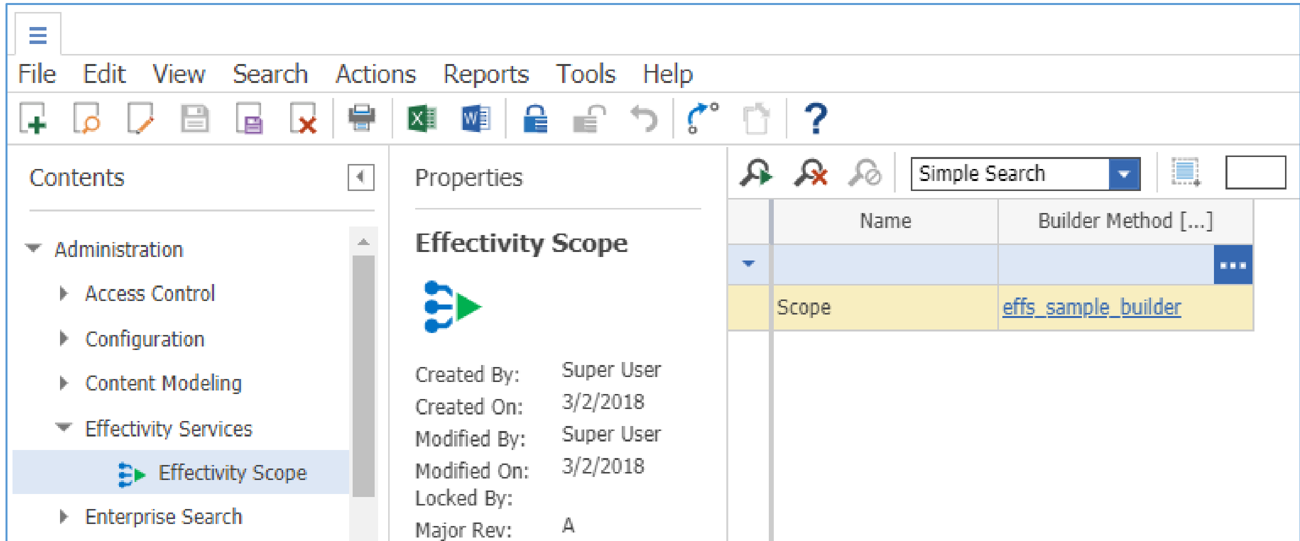


Figure 7.

The Effectivity Scope (effs_scope) is related to the Effectivity Scope ItemType (effs_scope_itemtype). The related_id links to the Relationship ItemType which is managed by effectivity.

Note: In this guide, the Part BOM ItemType is used as an example to manage effectivity.

2. Double-click **Scope** to access the Scope Item.
3. Click the **New Relationship** icon on the Effectivity Scope ItemType tab. The Itemtype Search dialog box appears.

©2018 Aras Corporation All Copyrights Reserved.

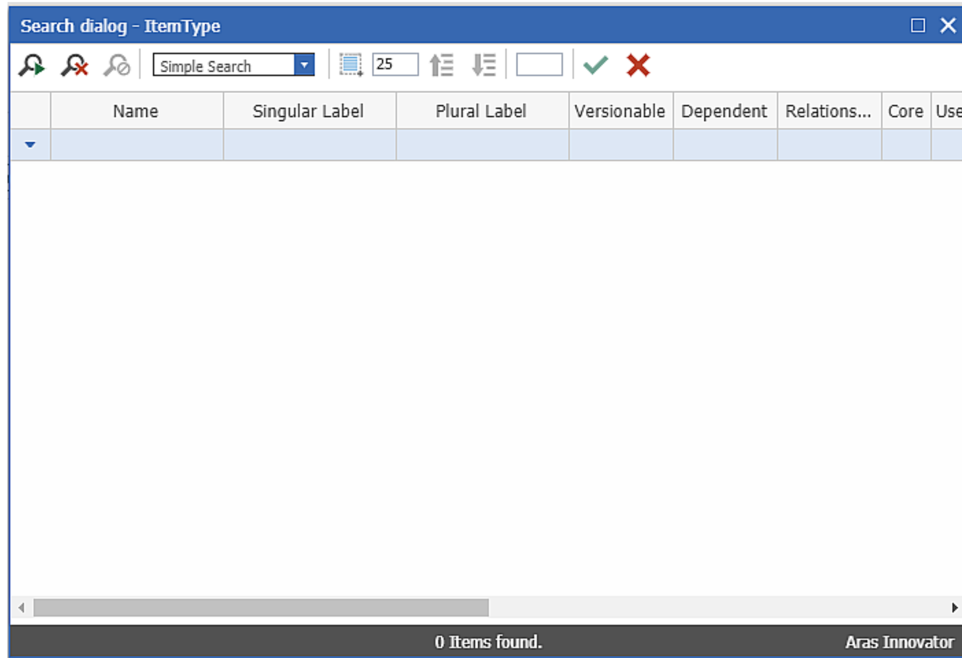


Figure 8.

4. Search for Part BOM and double click on it to add it to the Effectivity Scope ItemType.
5. Save the current Effectivity Scope.

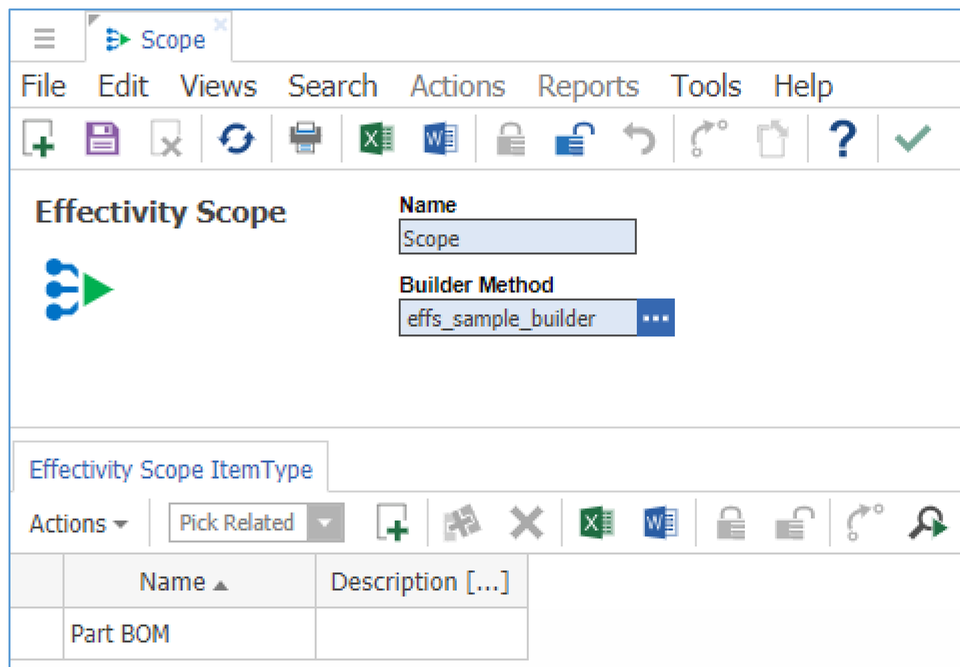


Figure 9.

Note: You can have any number of different Relationships in the same Effectivity Scope.

©2018 Aras Corporation All Copyrights Reserved.

New "Effectivity" (effs_Part_BOM_expression) "No Related" relationship ItemType will be automatically generated. The Source_id of effs_Part_BOM_expression is a link to Part BOM, and the related_id is null. With this configuration, the "Effectivity" tab appears on the Part BOM form.

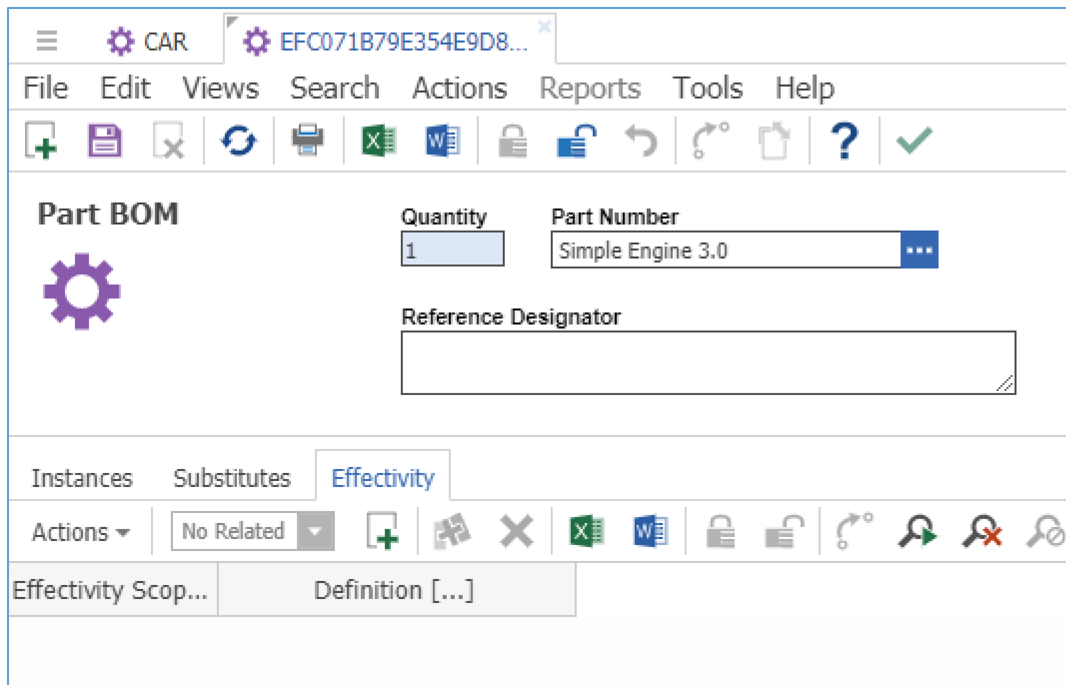
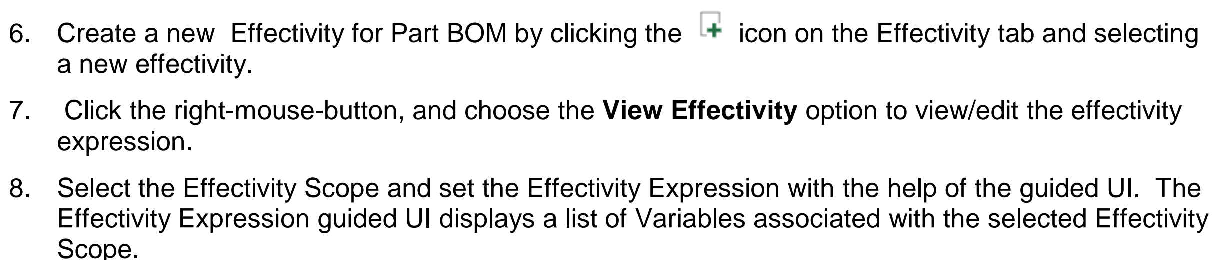


Figure 10.

Note:	Log out of Aras Innovator and then log in as new relationship ItemTypes are created automatically, unless they already exist.



6. Create a new Effectivity for Part BOM by clicking the icon on the Effectivity tab and selecting a new effectivity.
7. Click the right-mouse-button, and choose the **View Effectivity** option to view/edit the effectivity expression.
8. Select the Effectivity Scope and set the Effectivity Expression with the help of the guided UI. The Effectivity Expression guided UI displays a list of Variables associated with the selected Effectivity Scope.

©2018 Aras Corporation All Copyrights Reserved.

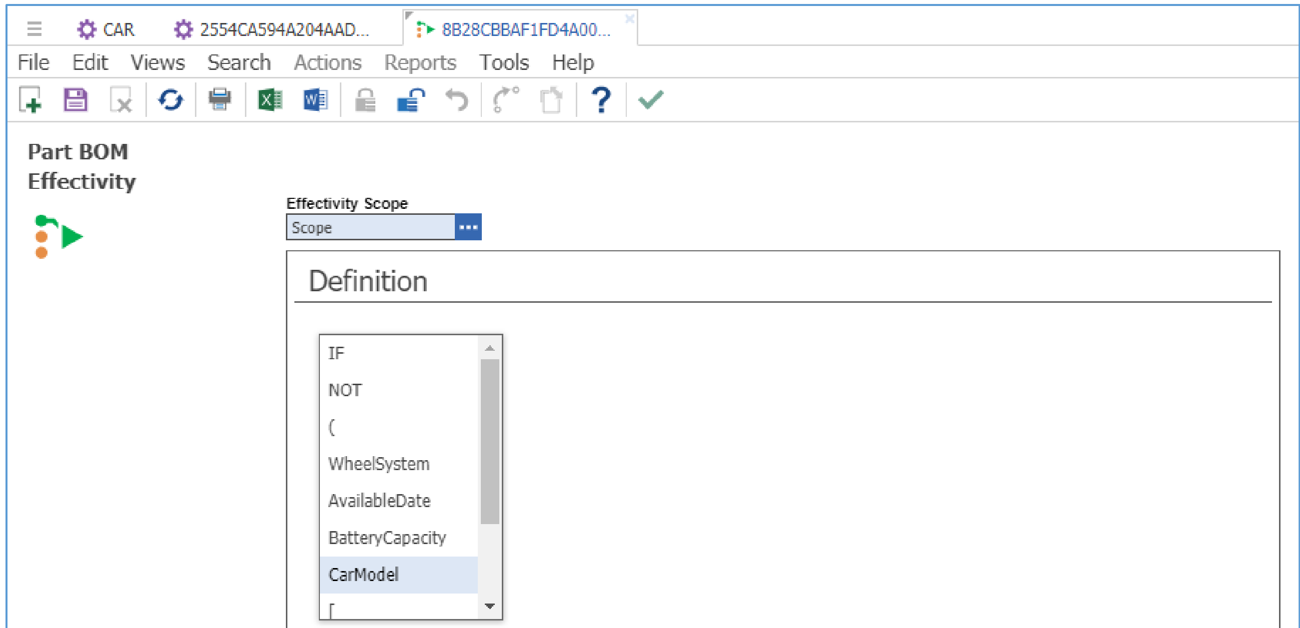


Figure 11.

©2018 Aras Corporation All Copyrights Reserved.

9 API Usage

9.1 Overview

Using Query Builder with Effectivity Services resolved item structures can be presented in a Tree Grid View. For additional information about Query Builder please refer to the "Aras Innovator 11.0 – Query Builder Guide." For additional information about Tree Grid View, refer to the "Aras Innovator 11.0 – Tree Grid View Administrator Guide." Both Guides are distributed with the Aras Innovator CD image.

This section describes the following use cases using Effectivity Services combined with Query Builder and Tree Grid View:

- Resolving a Part BOM structure for given effectivity criteria.
- Viewing all parts in a Part BOM structure that are a) effective for the criteria contained in the specified Effectivity Scope and b) have no Effectivity Expressions in the specified Effectivity Scope.

Before you can configure a Query Definition and a Tree Grid View, you need to:

- Define business data that will be converted to Variables and Named Constants by the Builder Method.
- Implement the Builder Method.
- Create the Effectivity Scope.
- Relate a Relationship ItemType to the Effectivity Scope instance.

To follow the example in this section the Administrator has already defined the following structural elements, shown in Figure 12:

- The Effectivity Scope item "Car Scope". The Effectivity Scope item has the Relationship ItemType "Part BOM" as the Effectivity Scope ItemType.
- The Effectivity Variables associated with the Effectivity Scope item:
  - WheelSystem – is an Enum Variable. That means this Effectivity Variable has an enumerated list of possible Named Constants (values / options – Rear, Front, Four), which can be assigned to it.
  - BatteryCapacity – is an Int Variable. That means any appropriate integer value can be assigned to this Effectivity Variable.
  - CarModel – is a String Variable. That means any appropriate string value can be assigned to this Effectivity Variable.
  - AvailableDate – is a DateTime Variable. That means any appropriate date value can be assigned to this Effectivity Variable.

©2018 Aras Corporation All Copyrights Reserved.

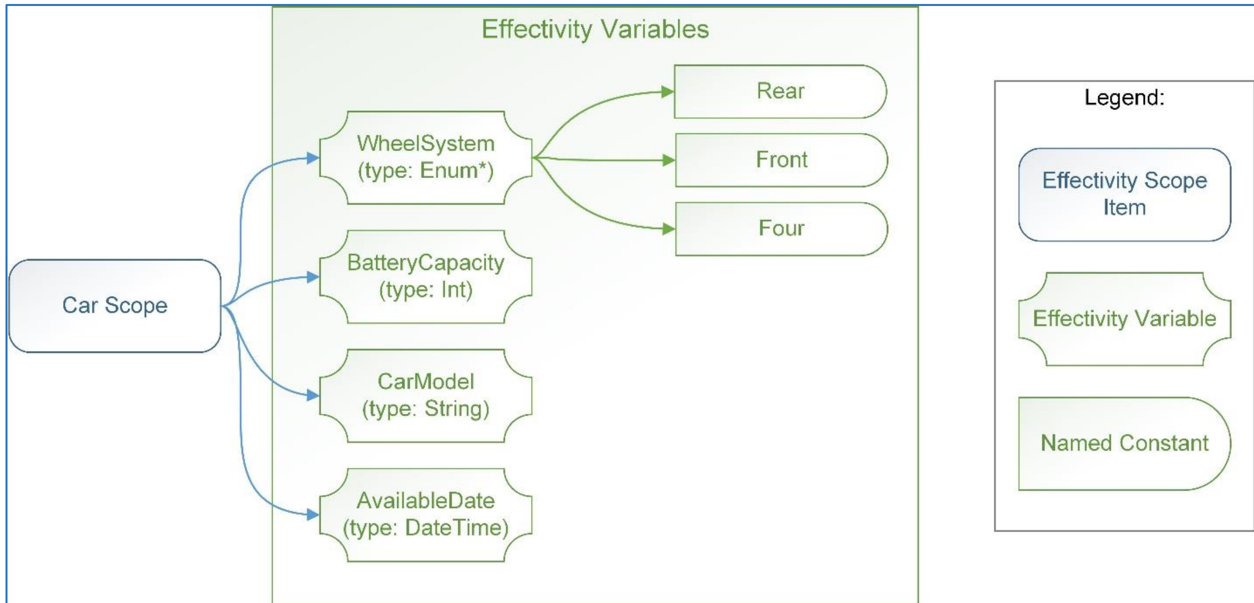


Figure 12.

9.2 Creating a Query Definition to Filter by Effectivity

A Query Definition is a fundamental element which we use to retrieve data from the server.
In this section we will use the **Part -> Part BOM** structure to show how Effectivity Services can be used.

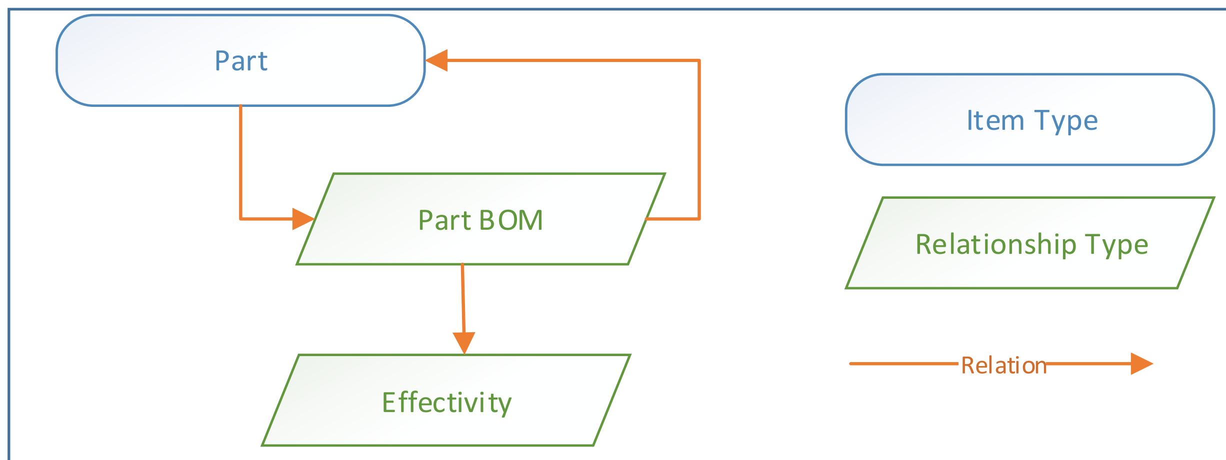


Figure 13.

©2018 Aras Corporation All Copyrights Reserved.

9.2.1 Setting up a Query Definition

9.2.1.1 Creating a Query Definition

Go to **Administration (1) / Configuration (2) / Query Definition (3)** in the TOC.

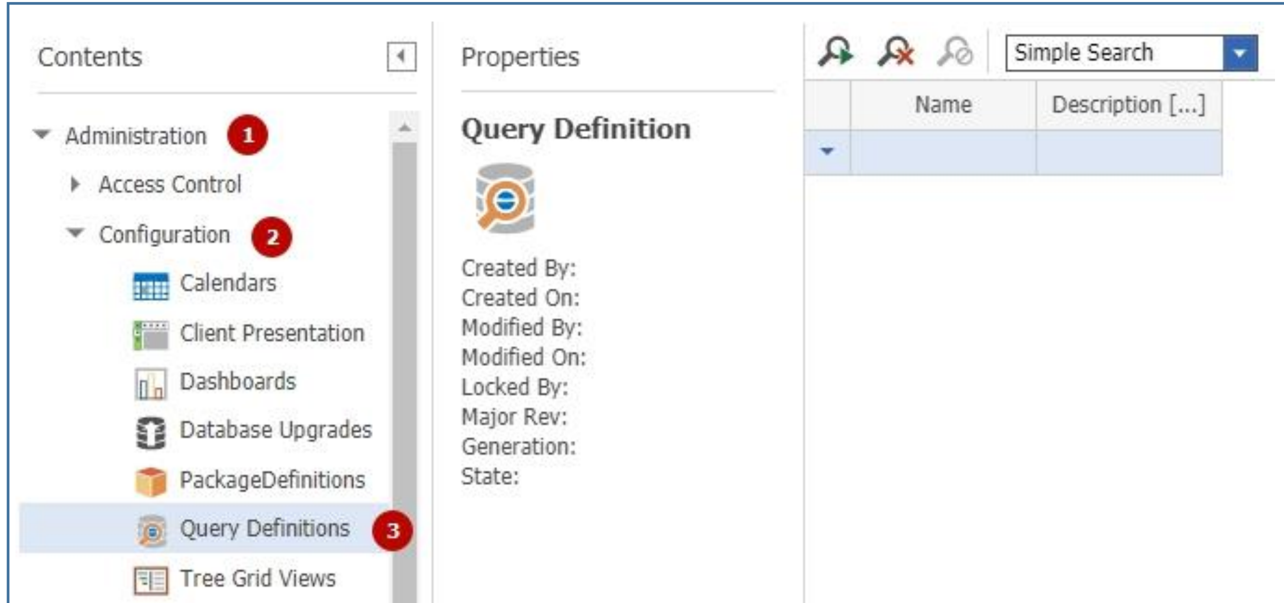


Figure 14.

9.2.1.2 Selecting Context Item Type Query Definition

1. Create an item and enter a value in the **Name** field (1).
2. Click on the ellipses in the Context ItemType dialog to search for a **Context Item Type** (2). Once you select the ItemType and save the form, the selected ItemType appears as a link as shown here.

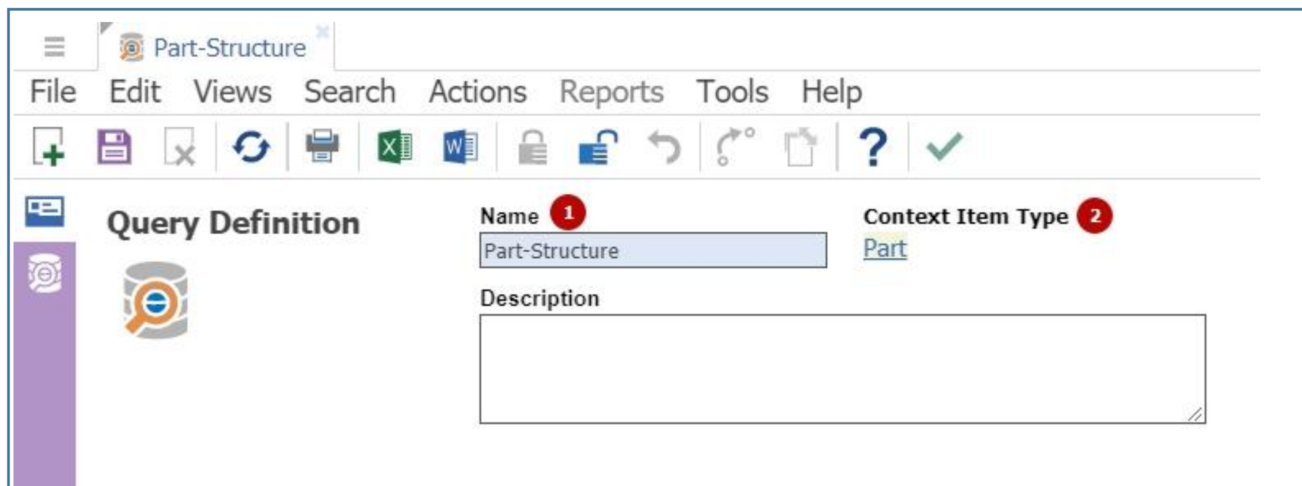


Figure 15.

9.2.1.3 Building a Recursive Part Structure

Click on the **Show Editor** button on the left sidebar to switch to Query-Editing Mode. Build a recursive part structure using the Part BOM relationship.

©2018 Aras Corporation All Copyrights Reserved.

aras

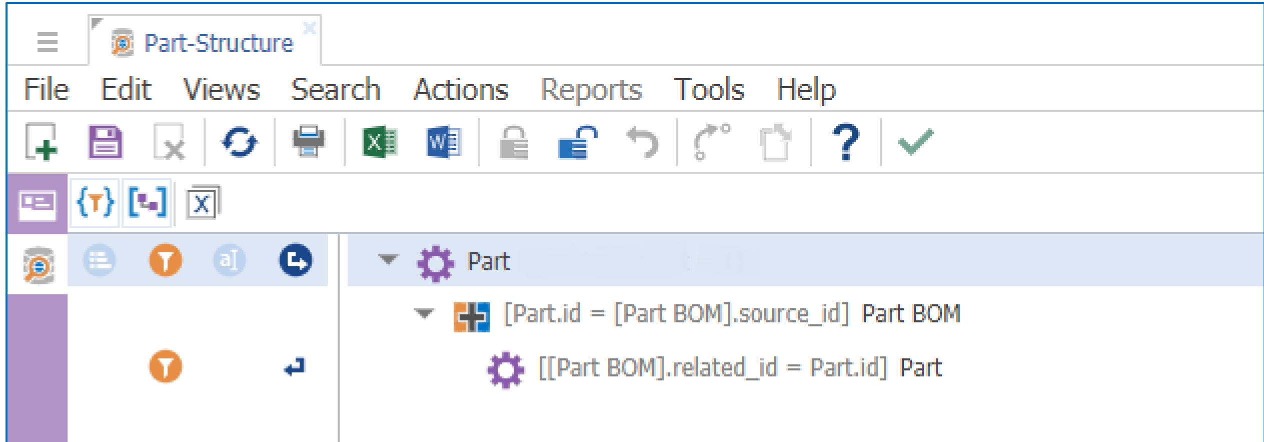


Figure 16.

9.2.1.4 Selecting Properties

Select the properties that you want the query to return. Let's select the following properties for the Part:

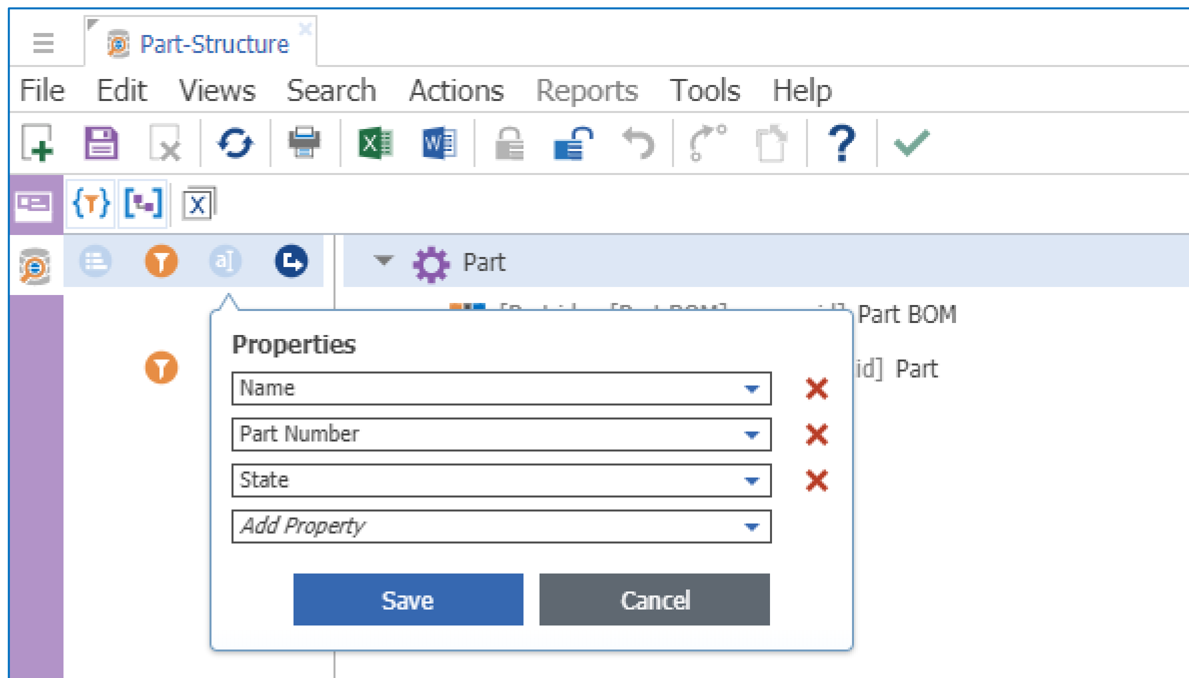


Figure 17.

9.2.2 Defining Effectivity Criteria

In order to define effectivity criteria, add the corresponding "Effectivity" relationship (effs_Part_BOM_expression), where Effectivity Conditions are saved.

©2018 Aras Corporation All Copyrights Reserved.

9.2.2.1 Adding two "Effectivity" relationships to the "Part BOM" item

Figure 18.

1. Add the **Effectivity** relationship twice with different aliases as shown in [Figure 19.](#)
2. Use the Relationship with the **effs_Part_BOM_expression** alias for structure resolution by effectivity.
3. Use the Relationship with the **effs_Part_BOM_expr_no_resolve** alias to check if the Part BOM item has any "Effectivity" relationship Items for the specified Scope.

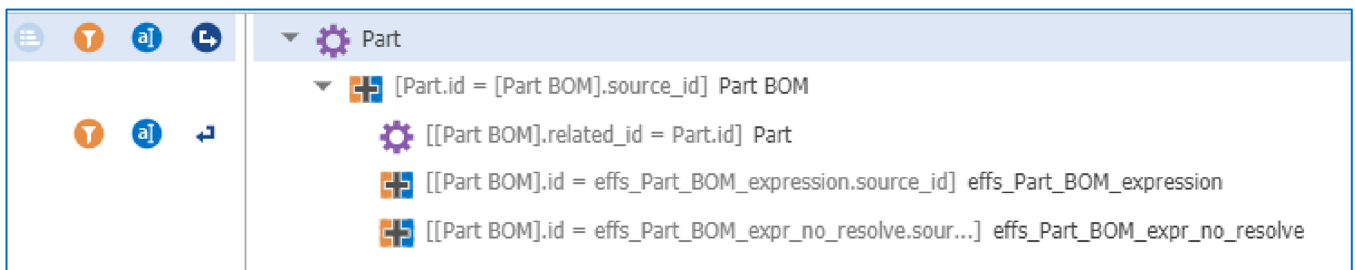


Figure 19.

©2018 Aras Corporation All Copyrights Reserved.

aras

These two aliases are used in the where condition on the **Part BOM** query item.

9.2.2.2 Adding a Part BOM where condition

Add a where condition to the "Part BOM" query item. The where condition shown in Figure 20 allows us to get the related part in case the Part BOM item does not have any effectivity relationships (effs_Part_BOM_expr_no_resolve = 0), or it has at least one effectivity expression resolved to true (effs_Part_BOM_expression > 0) for the given effectivity resolution criteria/condition.

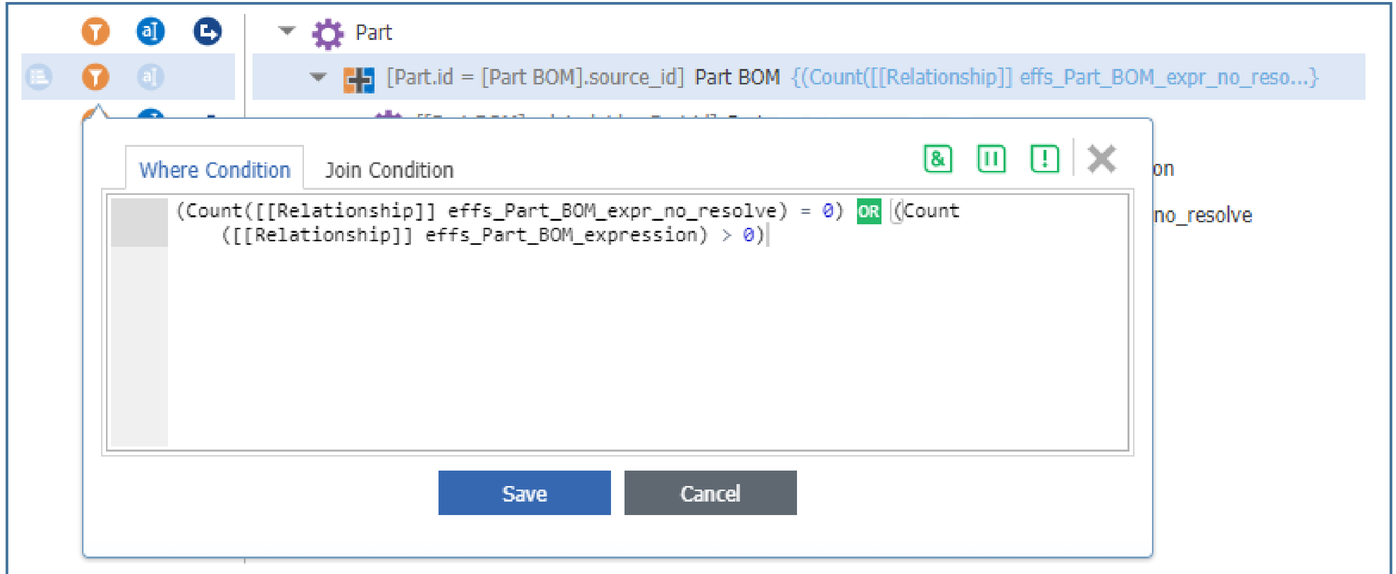


Figure 20.

9.2.2.3 Adding an effs_Part_BOM_expr_no_resolve where condition

Add a where condition on the "effs_Part_BOM_expr_no_resolve" query item filtering by the "Effectivity Scope" and "behavior" properties. The "Effectivity Scope" value must match the ID of the scope created in the **Administration / Effectivity Services / Effectivity Scope TOC** category.

This gives us Part BOMs that have no configured Effectivity Expression in the current Scope.

Note: The "(behavior = 'float' OR behavior = 'fixed')" condition is required in order to filter the search using the latest version of the relationship item.

©2018 Aras Corporation All Copyrights Reserved.

aras

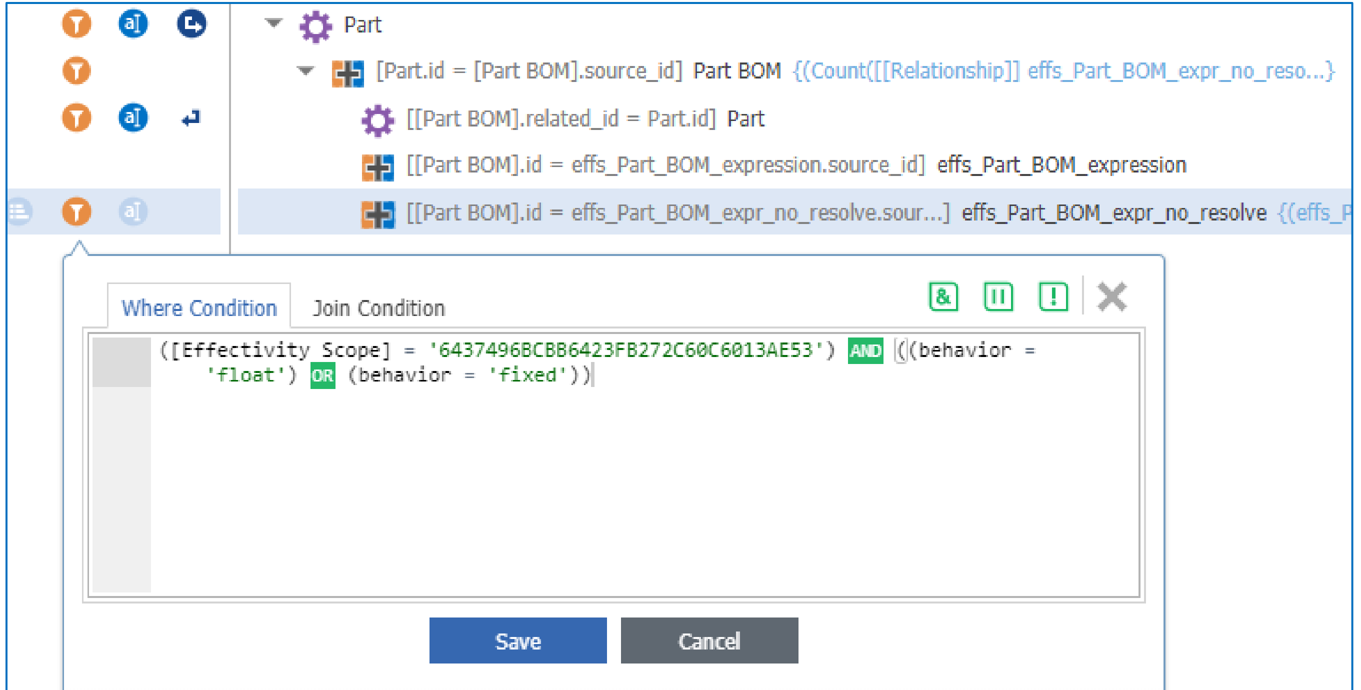


Figure 21.

9.2.2.4 Adding an "effs_Part_BOM_expression" where condition

Add a where condition on the "effs_Part_BOM_expression" query item filtering by the "Effectivity Scope," "behavior," and "definition" properties.

The "Effectivity Scope" value must match the ID of the scope created in the **Administration / Effectivity Services / Effectivity Scope TOC** category.

The Definition property value must contain an effectivity expression that will be used for resolving the part structure.

This gives us Part BOMs that have a configured Effectivity Expression in the current Scope. The configured Effectivity Expression corresponds to the Effectivity criteria from the "definition" property of the Query Item.

Note: The "(behavior = 'float' OR behavior = 'fixed')" condition is required to filter the search using the latest version of the relationship item.

In the following figure the "Definition" property contains an effectivity expression with a $WheelSystem query parameter.

©2018 Aras Corporation All Copyrights Reserved.

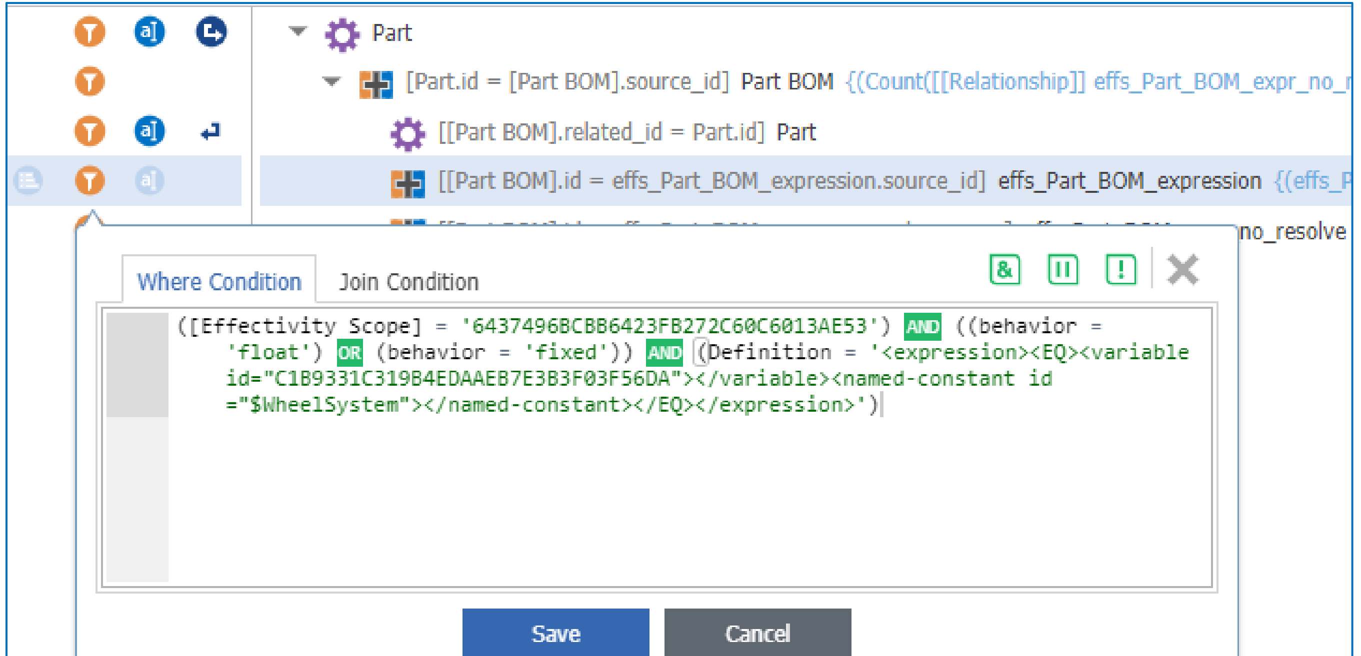


Figure 22.

Note: Only <EQ> is supported in effectivity resolution criteria. Only one Effectivity Scope can be used in a "Where Condition."

Query parameters allow us to execute the same query using different parameter values.

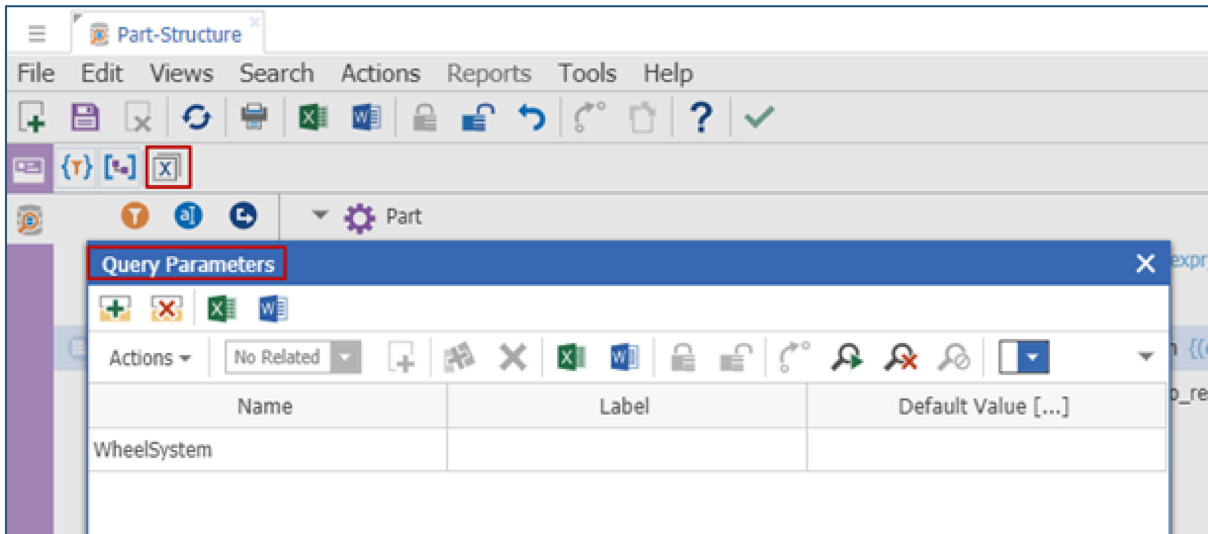


Figure 23.

9.2.2.5 *Confirming Conditions*

Make sure all conditions are set correctly. The final version of the effectivity query definition should look like this:

©2018 Aras Corporation All Copyrights Reserved.

aras

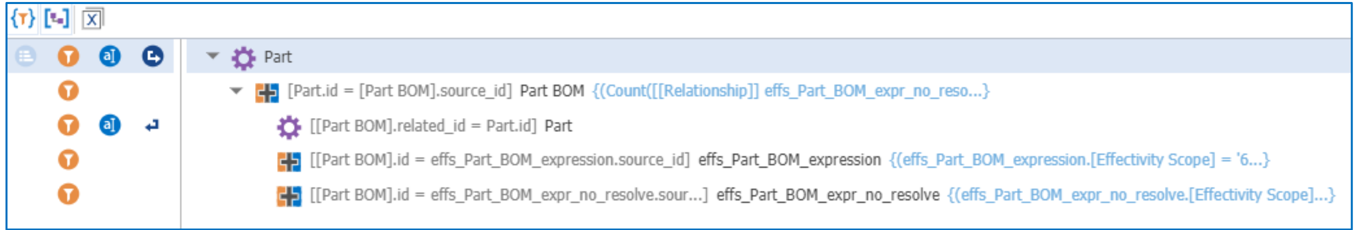


Figure 24.

9.3 Creating a Tree Grid View to Display Effective Items

Tree Grid View is standard functionality for Aras Innovator. It offers a visual layout of the data as a Relationship tab in an item view. In this example Tree Grid View uses the Query Definition created in the previous section to resolve the Part BOM structure and display the result in the grid.

9.3.1 Setting Up a Tree Grid View

9.3.1.1 Tree Grid Views

Select **Administration / Configuration / Tree Grid Views** in the TOC.

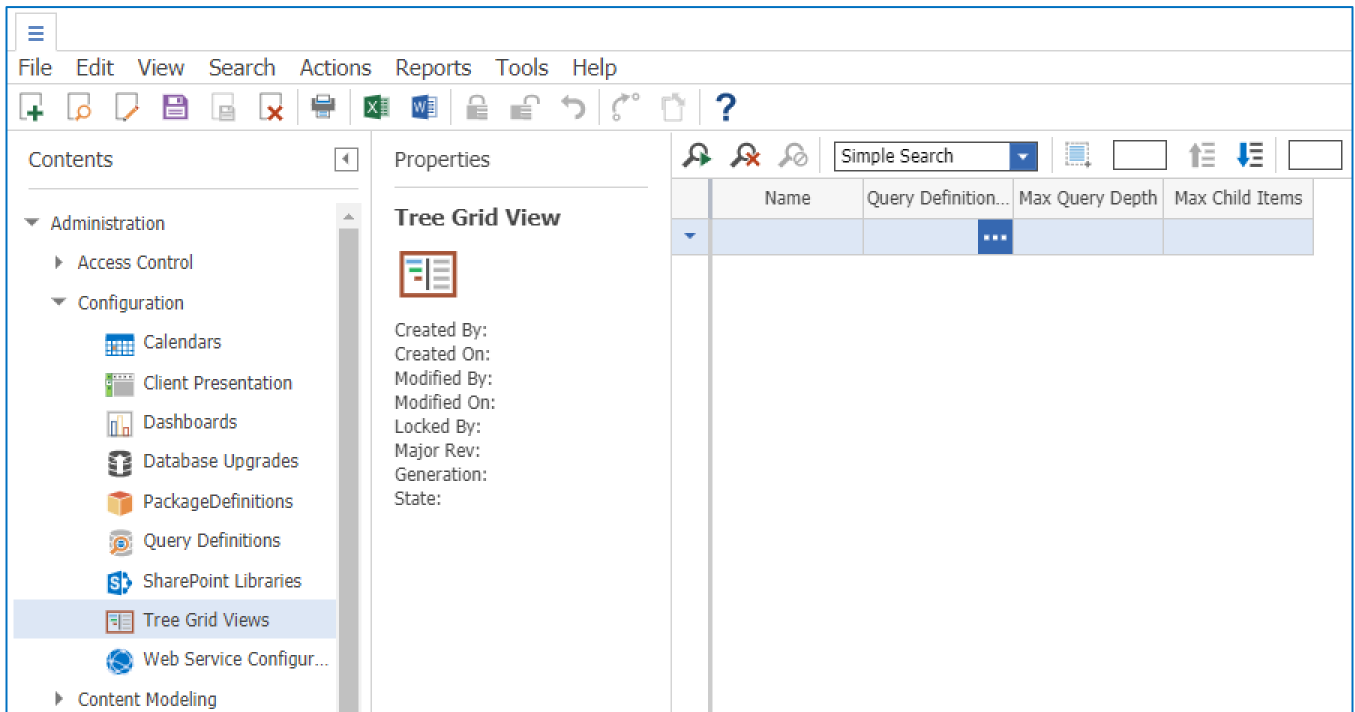


Figure 25.

9.3.1.2 Creating a Tree Grid View Item

Create a Tree Grid View item. Set the required "Name" property and search for a query definition in the "Query Definition" field.

©2018 Aras Corporation All Copyrights Reserved.

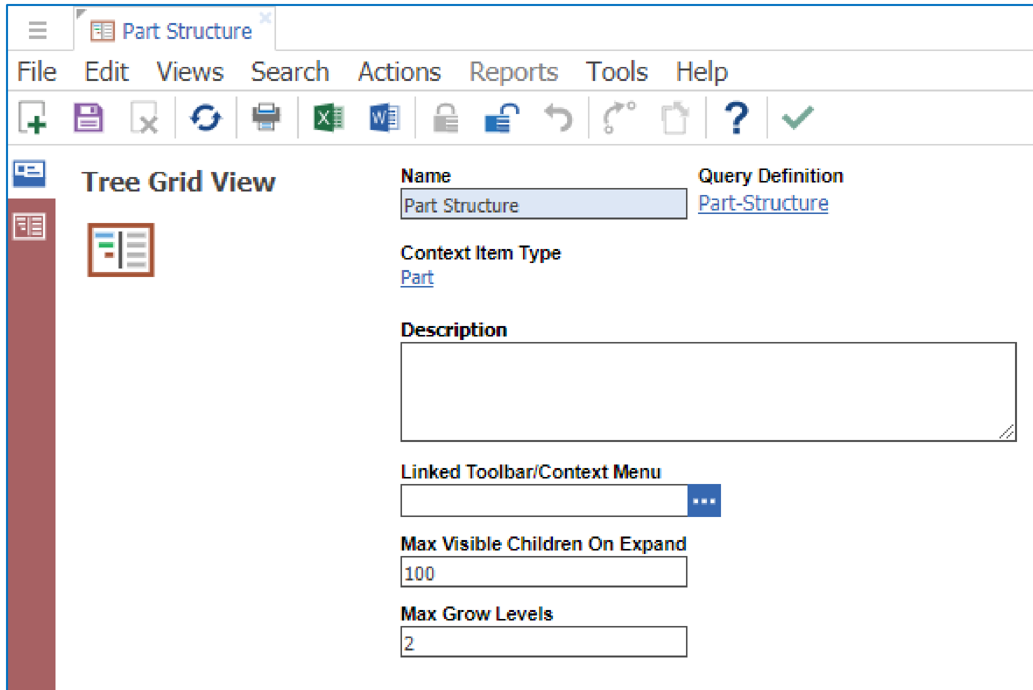


Figure 26.

9.3.1.3 Using the Tree Grid View Editor

Click the **Show Editor** button on the sidebar.

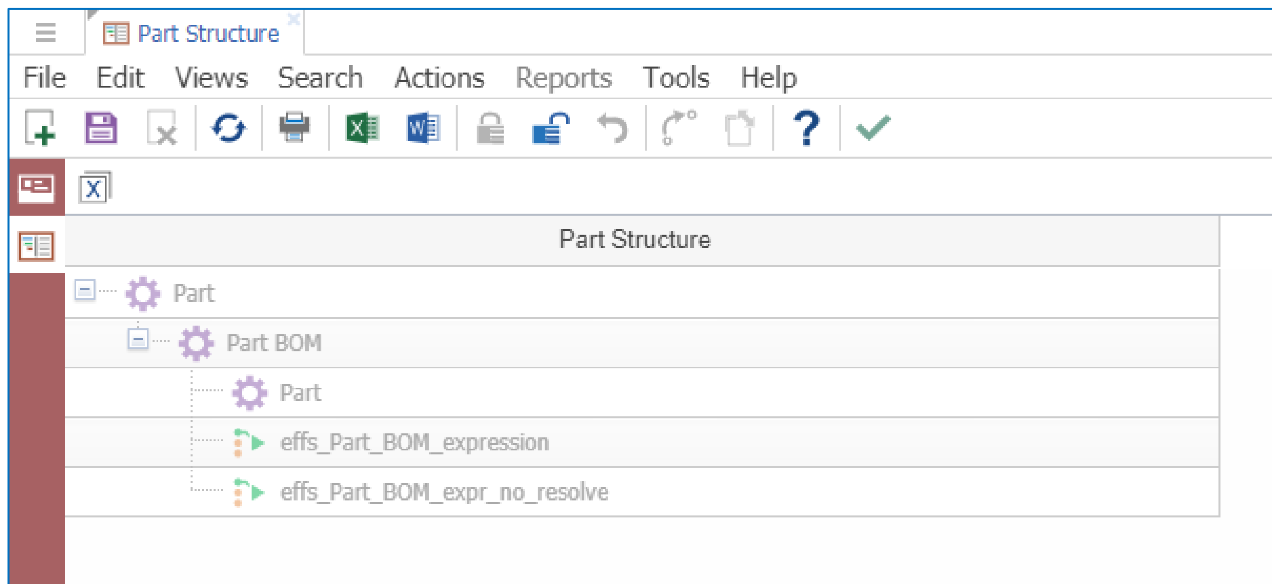


Figure 27.

9.3.1.4 Mapping Elements

Select the "Part," "Part BOM," and "Part" rows, right click on them and choose **Map Element** from the context menu to include these elements in the tree grid view.

©2018 Aras Corporation All Copyrights Reserved.

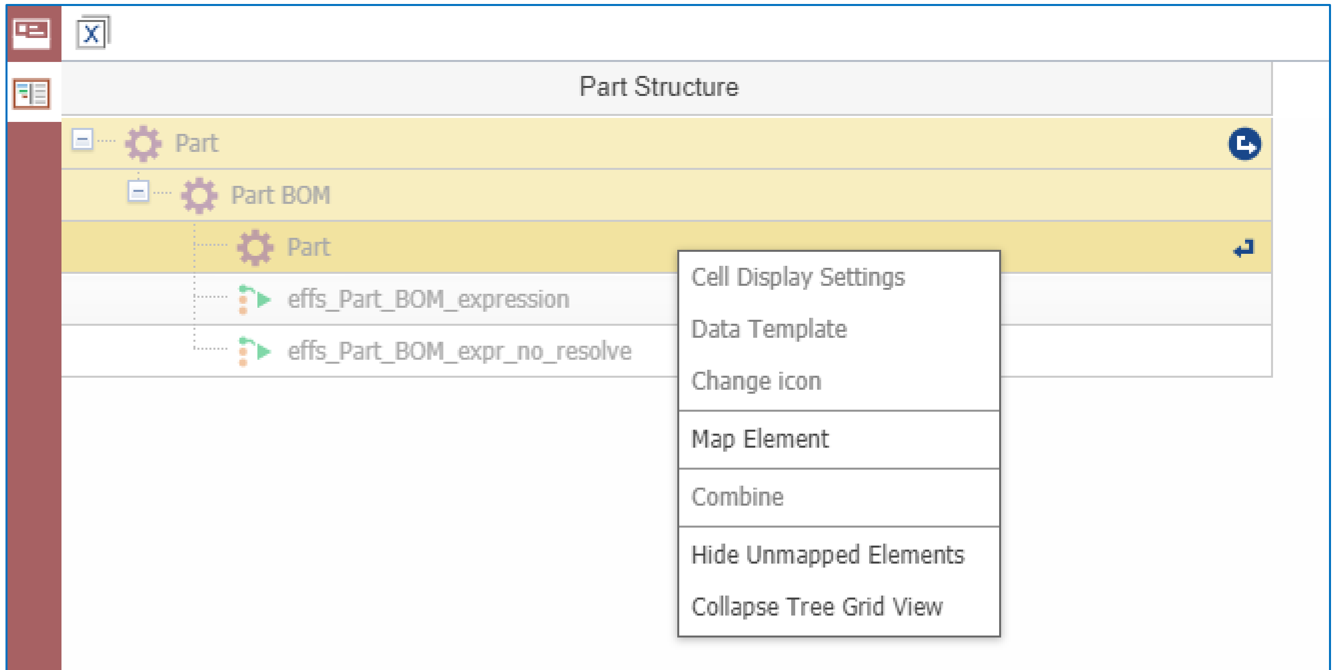


Figure 28.

9.3.1.5 Combining Elements

Select Part BOM and related Part elements, right click on them and choose **Combine** from the context menu to combine them into one element for better structure visualization in the Tree Grid View.

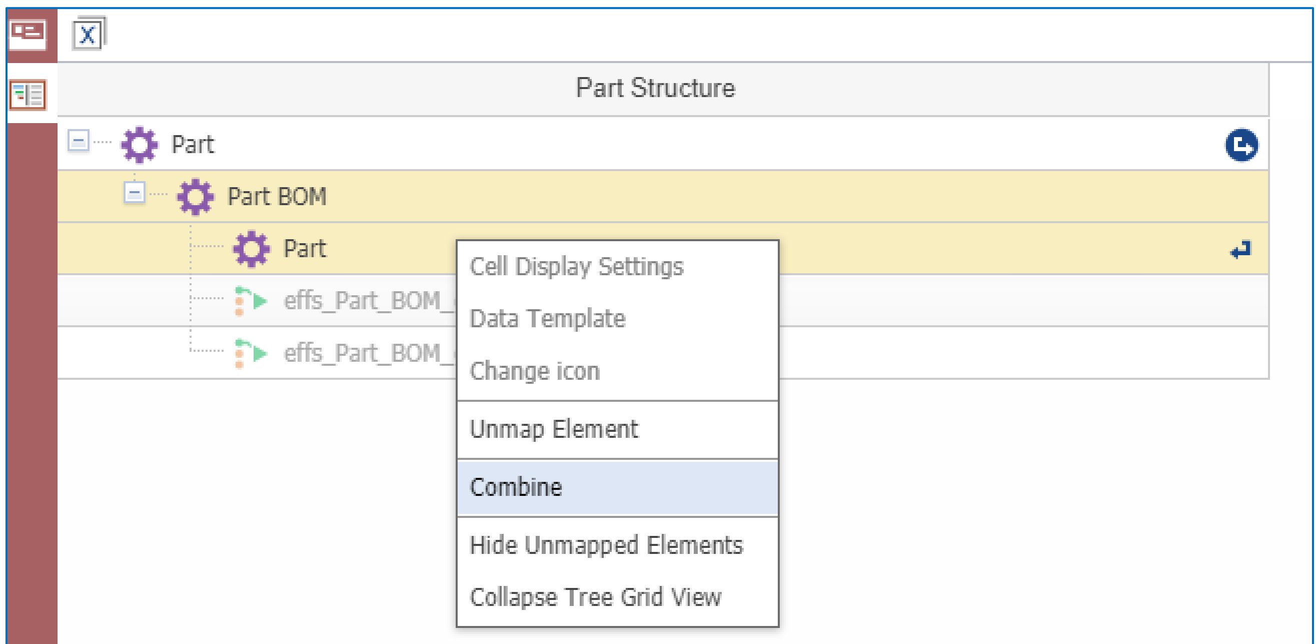


Figure 29.

9.3.1.6 Adding Columns

Right click on the **Part Structure** column header and select **Add New Column**.

©2018 Aras Corporation All Copyrights Reserved.

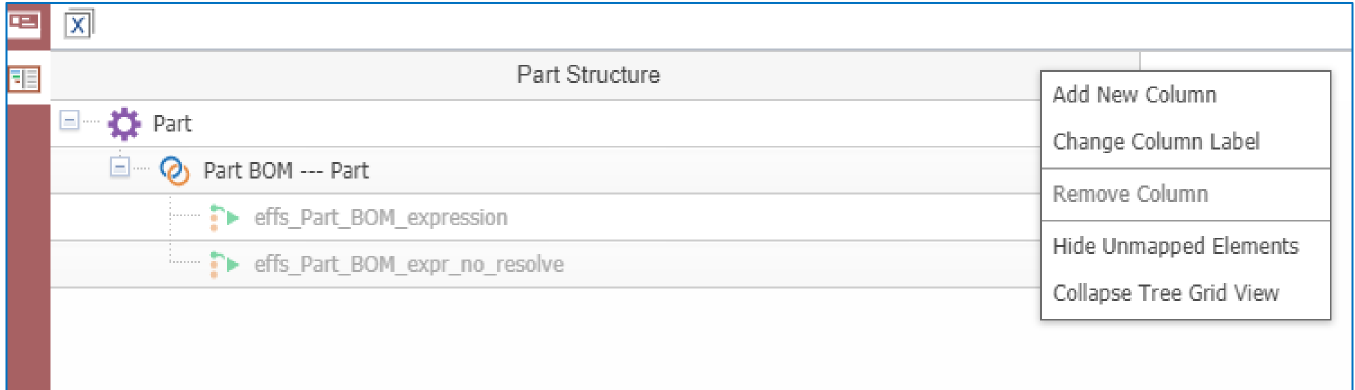


Figure 30.

9.3.1.7 *Changing the Column Label*

Change new column label to **State**.

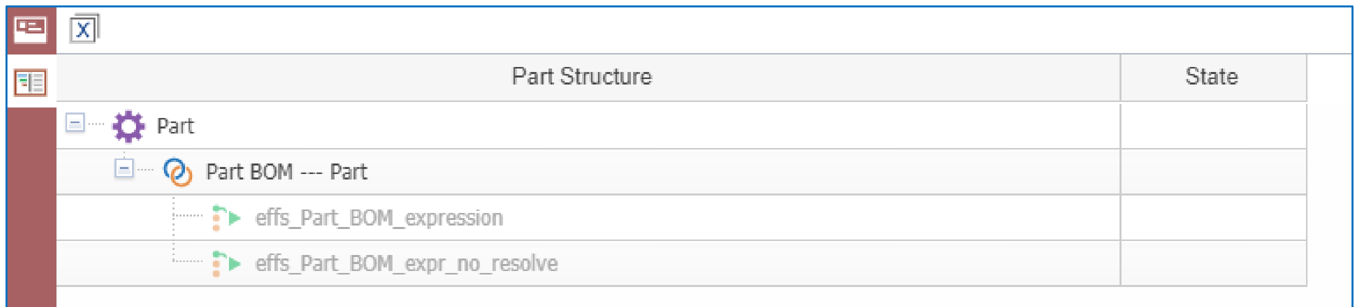


Figure 31.

9.3.1.8 *Choosing Cell Display Settings*

Select the **State** cell in the Part row, right click on it and choose **Cell Display Settings** from the context menu.

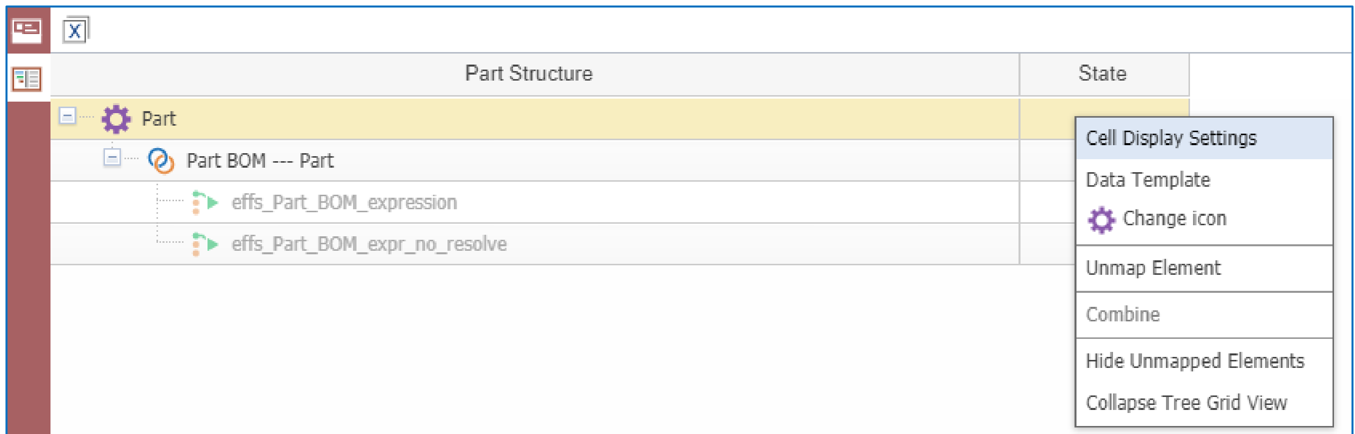


Figure 32.

9.3.1.9 *Setting text templates*

1. Select the **{Part.state}** option in the Helper listbox to set it as the text template for this cell.

©2018 Aras Corporation All Copyrights Reserved.

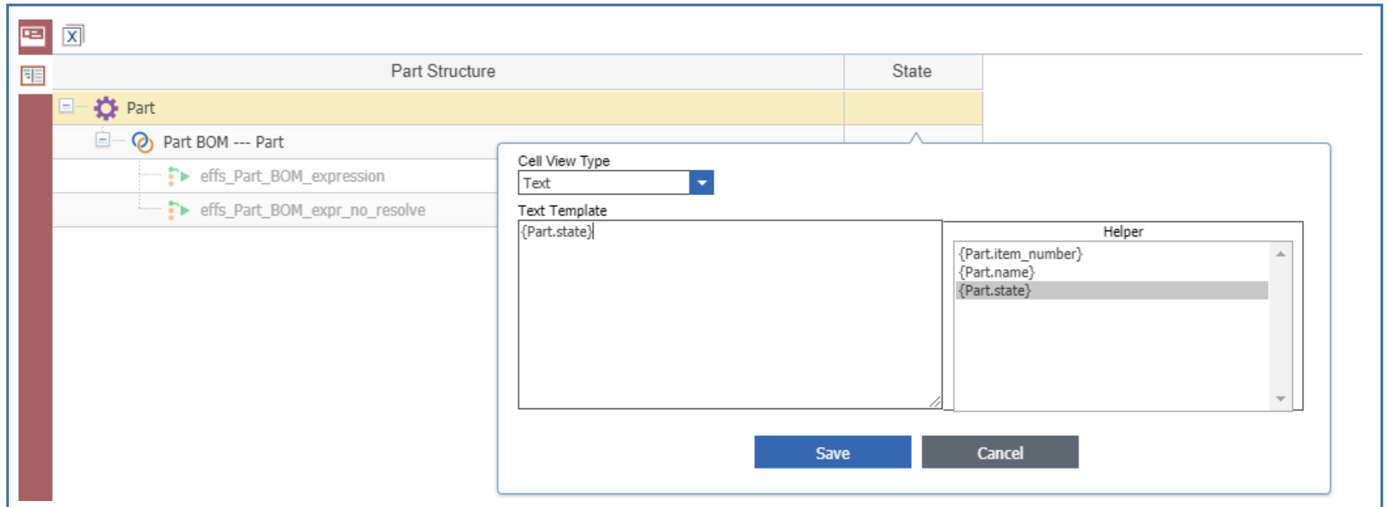


Figure 33.

2. Set **{Part.state}** as the text template for the "State" cell in the "Part BOM --- Part" row.

Part Structure	State
Part	{Part.state}
Part BOM --- Part	{Part.state}
effs_Part_BOM_expression	
effs_Part_BOM_expr_no_resolve	

Figure 34.

3. Add the **Part Number** column and setup display settings as shown here:

Part Structure	Part Number	State
Part {Part.name}	{Part.item_number}	{Part.state}
Part BOM --- Part {Part.name}	{Part.item_number}	{Part.state}
effs_Part_BOM_expression		
effs_Part_BOM_expr_no_resolve		

Figure 35.

9.3.1.10 Parameter Mapping

Click the **Show Parameter Mapping** button and map the existing query parameters to their data sources. In this example the named constants for WheelSystem enum are represented by the items of "WheelSystem" ItemType. So the "Data Type" property is set to "Item" and the corresponding ItemType is specified in the "Data Source" property.

©2018 Aras Corporation All Copyrights Reserved.

aras

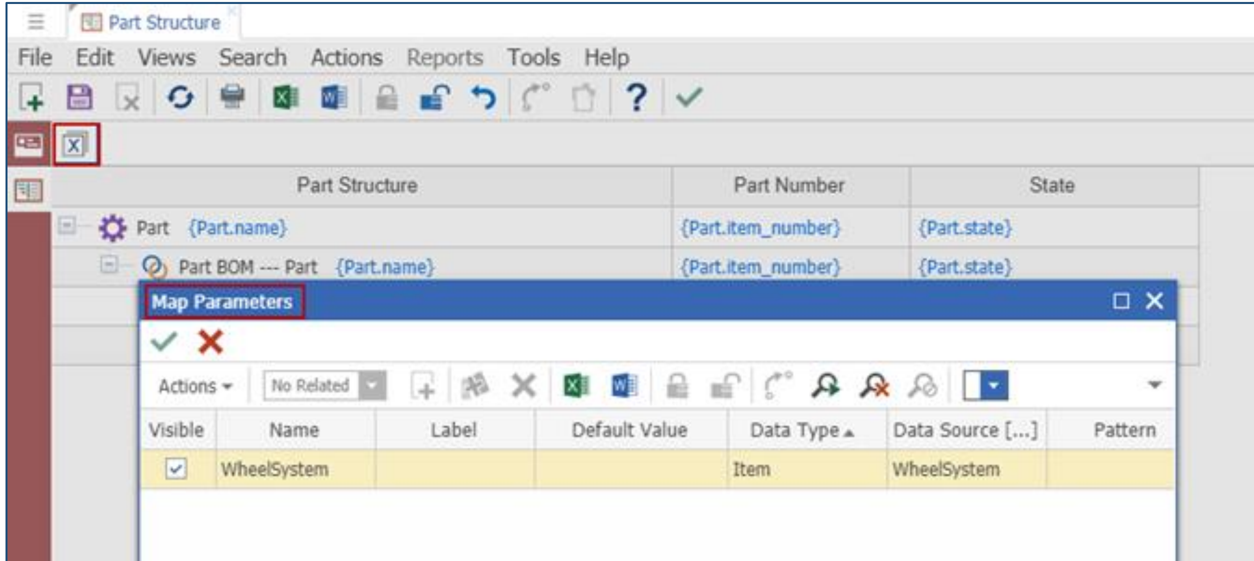


Figure 36.

9.3.1.11 Tree Grid View Usage

1. Click the Actions menu and select **Set Tree Grid View Usage**.

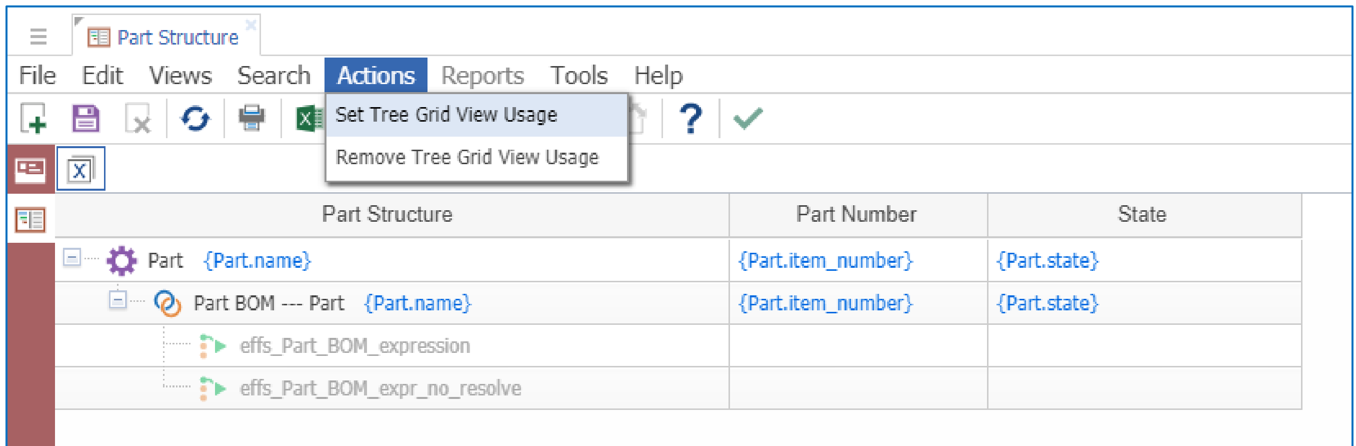


Figure 37.

2. Choose the **Relationship Tab** option as the target usage in the Set Tree Grid View Usage dialog box.

©2018 Aras Corporation All Copyrights Reserved.

aras

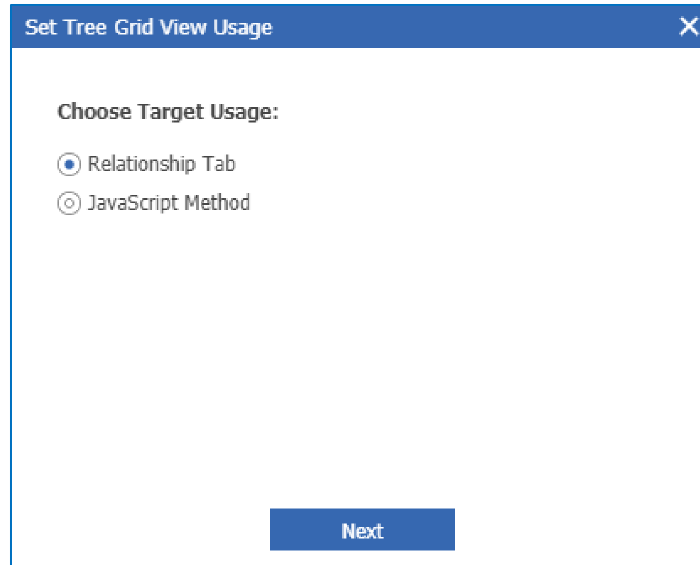


Figure 38.

3. Click the **Used On** button and select the ItemType to add to the new relationship.

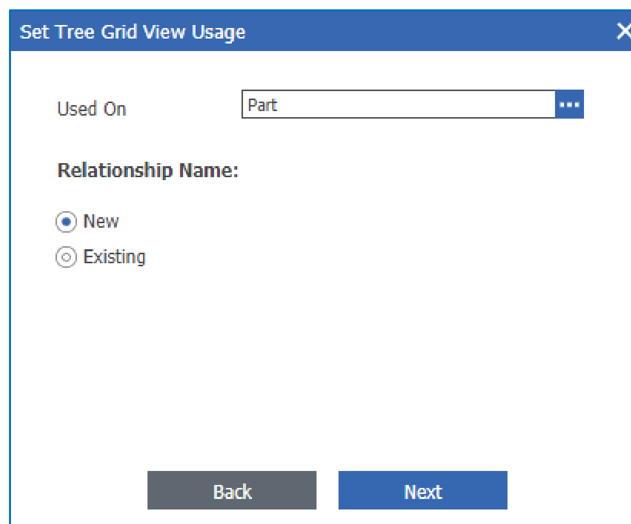


Figure 39.

4. Specify the relationship name and label.

©2018 Aras Corporation All Copyrights Reserved.

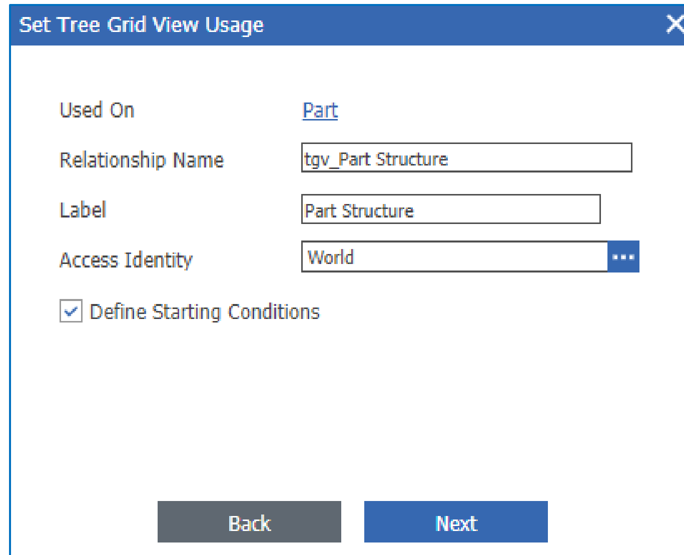


Figure 40.

5. Define the starting condition for query definition execution. The Starting condition is applied once to the root item of the query definition. It allows you to get the resolved part structure for one part item.

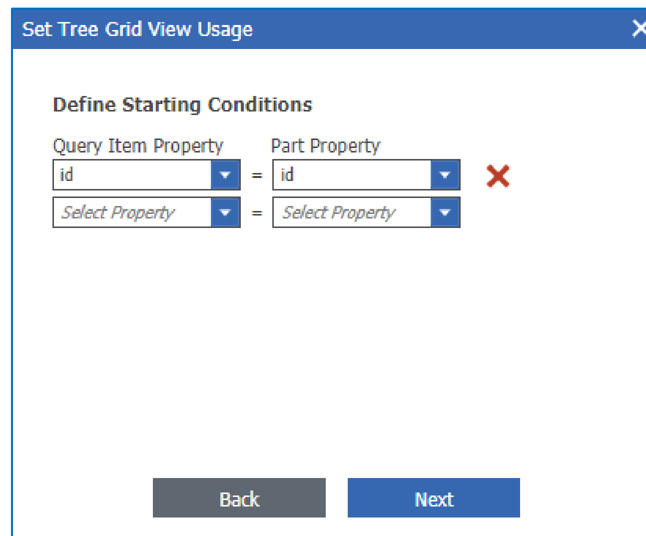


Figure 41.

6. Click **Generate** to create the relationship with the Tree Grid View.

©2018 Aras Corporation All Copyrights Reserved.

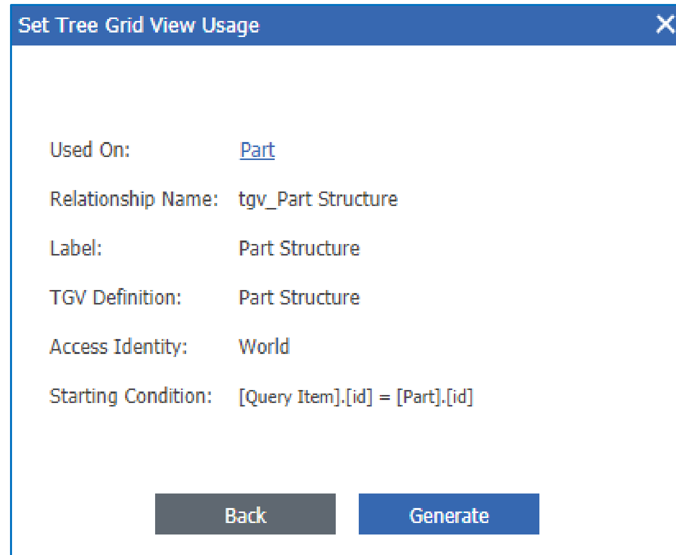


Figure 42.

9.3.2 Using Tree Grid View Parameters for Resolving Effective Items

The Resolved part structure for the current part item is displayed in the Tree Grid View. Figure 43 shows all of the related parts. The Effectivity expressions have not been added yet.

©2018 Aras Corporation All Copyrights Reserved.

CAR

File Edit Views Search Actions Reports Tools Help

Part

Created By: Super User
Created On: 3/2/2018
Modified By: Super User
Modified On: 3/5/2018
Locked By: Super User
Major Rev: A
Release Date:
Effective Date:
Generation: 2
State: Preliminary

Part Number: CAR
Revision: A
State: Preliminary
Name: CAR
Type
Unit: EA
Make / Buy: Make
Cost
Long Description
Assigned Creator
Designated User
Effective Date
Select an image...

Changes Pending

BOM | BOM Structure | Alternates | AML | Documents | CAD Documents | Goals | Changes | Part Submission Warrants | Part Structure

2

Part Structure	Part Number	State
CAR	CAR	Preliminary
Simple Engine 3.0	Simple Engine 3.0	Preliminary
Simple Engine 2.0	Simple Engine 2.0	Preliminary
Automatic GearBox	Automatic GearBox	Preliminary
Coupe	Coupe	Preliminary
Electric Engine 100A	Electric Engine 100A	Preliminary
Electric Engine 50A	Electric Engine 50A	Preliminary
Hybrid Engine	Hybrid Engine	Preliminary
Manual GearBox	Manual GearBox	Preliminary
Sedan	Sedan	Preliminary
Simple Engine 1.8	Simple Engine 1.8	Preliminary

Figure 43.

9.3.2.1 Using Tree Grid View parameters with Enum variables

In this section, we set effectivity expressions for Part BOM relationship items using the “Simple Engine 3.0” and “Simple Engine 2.0” related parts.

1. Go to the BOM tab, right click on part item, and select **View “BOM**.”

©2018 Aras Corporation All Copyrights Reserved.

BOM | BOM Structure | Alternates | AML | Documents | CAD Documents | Goals

Actions | Pick Related

Sequence	Part Number	Revision	Name
1	Simple Engine 3.0	A	Simple Engine 3.0
2	Simple		nple Engine 2.0
3	Autom		tomatic GearBox
4	Coupe		upe
5	Electric		ctric Engine 100A
6	Electric		ctric Engine 50A
7	Hybrid Engine	A	Hybrid Engine
8	Manual GearBox	A	Manual GearBox
9	Sedan	A	Sedan
10	Simple Engine 1.8	A	Simple Engine 1.8

View "Part"
View "BOM"
Pick/Replace "Part"
New
Delete

Figure 44.

2. Click the Effectivity tab and create a new relationship item. Right click on it and select View Effectivity.

Note: The Effectivity tab was added automatically when the "Part BOM" relationship ItemType was added to an Effectivity Scope item.

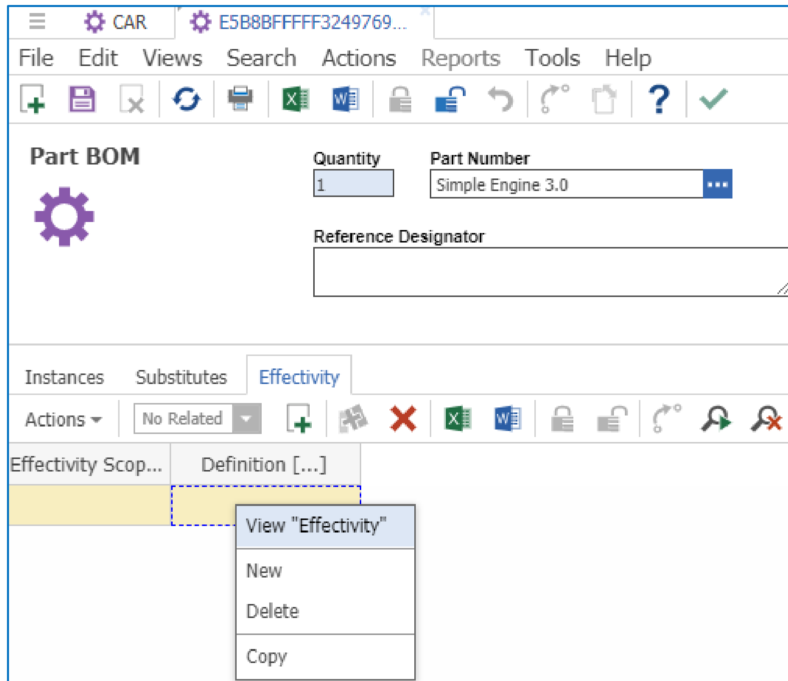


Figure 45.

3. Set the "Effectivity Scope" property, save the item, set the effectivity expression WheelSystem=Rear for Part BOM using "Simple Engine 3.0" as the related part. After that save "Part BOM Effectivity" item again, unlock and close it. Then save "Part BOM" item, unlock and close it.

©2018 Aras Corporation All Copyrights Reserved.

aras

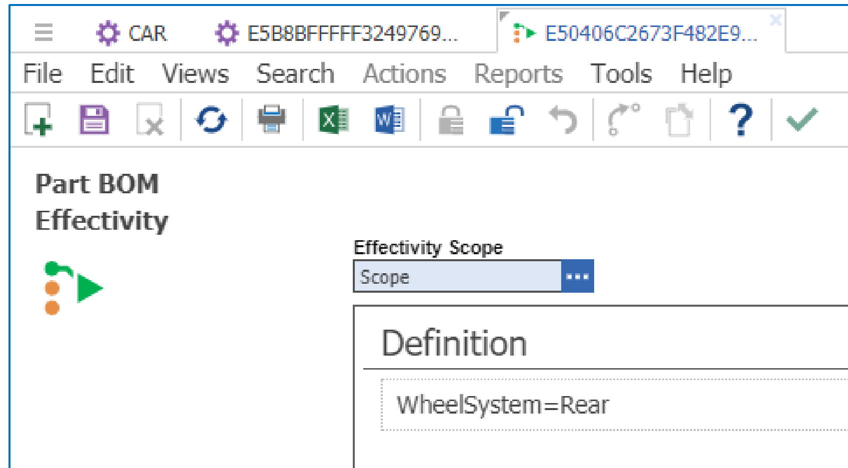


Figure 46.

4. Repeat the same steps to set the effectivity expression WheelSystem=Front for the Part BOM using "Simple Engine 2.0" as the related part.
5. Go to the **Part Structure** tab in the Tree Grid View and click the **Modify Parameters** button in the toolbar.
6. Set the **WheelSystem** query parameter value to named-constant "Four" represented by an item of the "WheelSystem" itemtype.

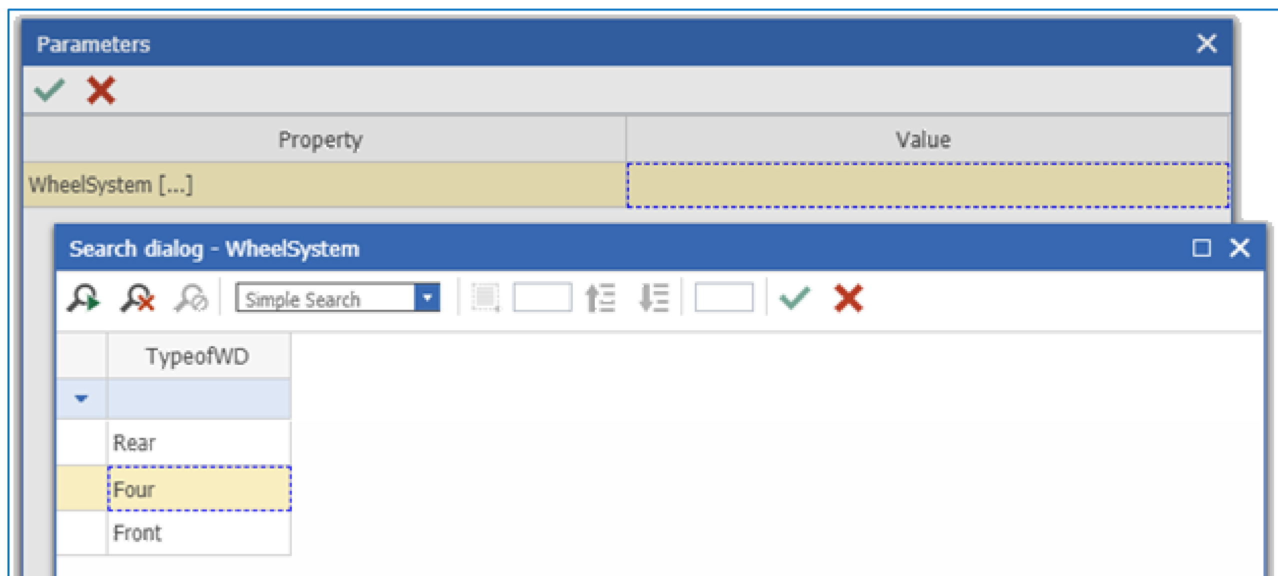


Figure 47.

©2018 Aras Corporation All Copyrights Reserved.

aras

"Simple Engine 3.0" and "Simple Engine 2.0" parts are not displayed in the Tree Grid View because their effectivity expressions conflict with the "WheelSystem=Four" expression for structure resolution.

BOM | BOM Structure | Alternates | AML | Documents | CAD Documents | Goals | Changes | Part Submission Warrants | Part Structure

Part Structure	Part Number	State
CAR	CAR	Preliminary
Automatic GearBox	Automatic GearBox	Preliminary
Coupe	Coupe	Preliminary
Electric Engine 100A	Electric Engine 100A	Preliminary
Electric Engine 50A	Electric Engine 50A	Preliminary
Hybrid Engine	Hybrid Engine	Preliminary
Manual GearBox	Manual GearBox	Preliminary
Sedan	Sedan	Preliminary
Simple Engine 1.8	Simple Engine 1.8	Preliminary

Figure 48.

7. Set the **WheelSystem** query parameter to "Rear" and the "Simple Engine 3.0" part is displayed.

BOM | BOM Structure | Alternates | AML | Documents | CAD Documents | Goals | Changes | Part Submission Warrants | Part Structure

Part Structure	Part Number	State
CAR	CAR	Preliminary
Simple Engine 3.0	Simple Engine 3.0	Preliminary
Automatic GearBox	Automatic GearBox	Preliminary
Coupe	Coupe	Preliminary
Electric Engine 100A	Electric Engine 100A	Preliminary
Electric Engine 50A	Electric Engine 50A	Preliminary
Hybrid Engine	Hybrid Engine	Preliminary
Manual GearBox	Manual GearBox	Preliminary
Sedan	Sedan	Preliminary
Simple Engine 1.8	Simple Engine 1.8	Preliminary

Figure 49.

8. Set the **WheelSystem** query parameter to "Front" and the "Simple Engine 2.0" part is displayed.

©2018 Aras Corporation All Copyrights Reserved.

aras

BOM | BOM Structure | Alternates | AML | Documents | CAD Documents | Goals | Changes | Part Submission Warrants | Part Structure

Part Structure	Part Number	State
CAR	CAR	Preliminary
Simple Engine 2.0	Simple Engine 2.0	Preliminary
Automatic GearBox	Automatic GearBox	Preliminary
Coupe	Coupe	Preliminary
Electric Engine 100A	Electric Engine 100A	Preliminary
Electric Engine 50A	Electric Engine 50A	Preliminary
Hybrid Engine	Hybrid Engine	Preliminary
Manual GearBox	Manual GearBox	Preliminary
Sedan	Sedan	Preliminary
Simple Engine 1.8	Simple Engine 1.8	Preliminary

Figure 50.

Note: You can also use the standard "List" ItemType to store named constants for Enum variables. In this case you must specify List as the data type and list item as the data source in the Tree Grid View query parameters mapping. You must set Named constant IDs in the "Value" properties as shown in Figure 51.

CAR | CarBodyColor

File Edit Views Search Actions Reports Tools Help

List

Name: CarBodyColor

Description

Value | Filter Value

Actions | No Related | Hide

Label	Value	Sort Order
Black	8B5EEF4FDE7941A3AA36E5053D468F2E	128
White	0C03BE365ECA4ED2825BA78780A5F74C	256
Gray	0E4FB1A47E9D442D98F35956B563DF31	384

Figure 51.

9.3.2.2 Using Tree Grid View parameters with Integer variables

This example uses the Query Builder and Tree Grid View parameters with the "BatteryCapacity" integer variable.

©2018 Aras Corporation All Copyrights Reserved.

aras

1. Create the effectivity expression **BatteryCapacity >= 100** on the "Part BOM" item with the "Coupe" related part.

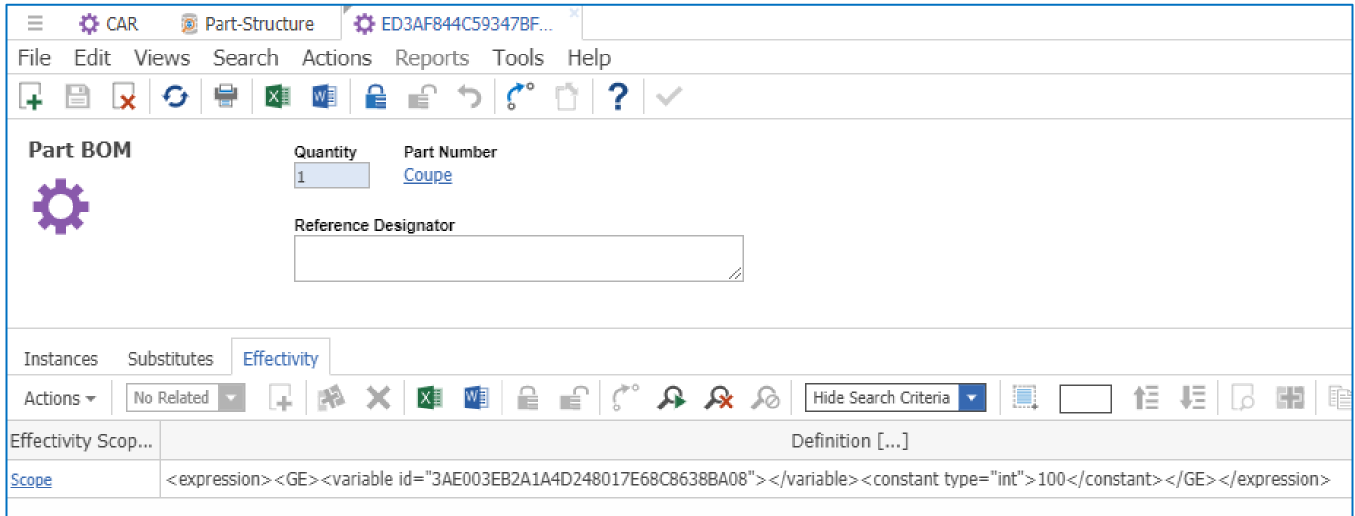


Figure 52.

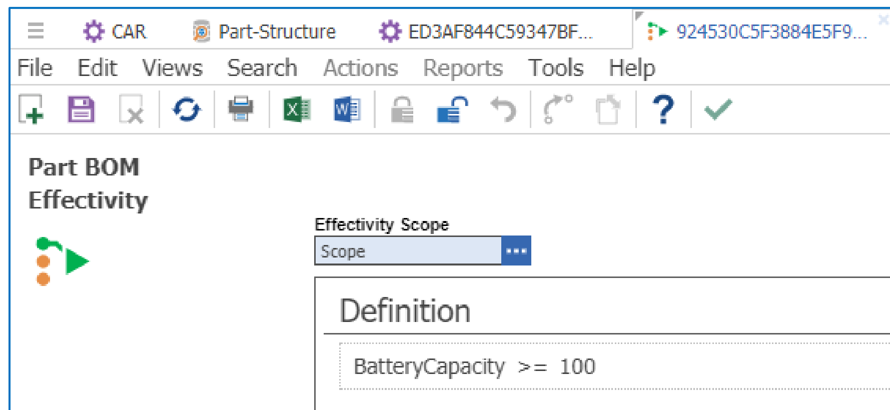


Figure 53.

2. Add the new query parameter **BatteryCapacity** in the query definition item.

©2018 Aras Corporation All Copyrights Reserved.

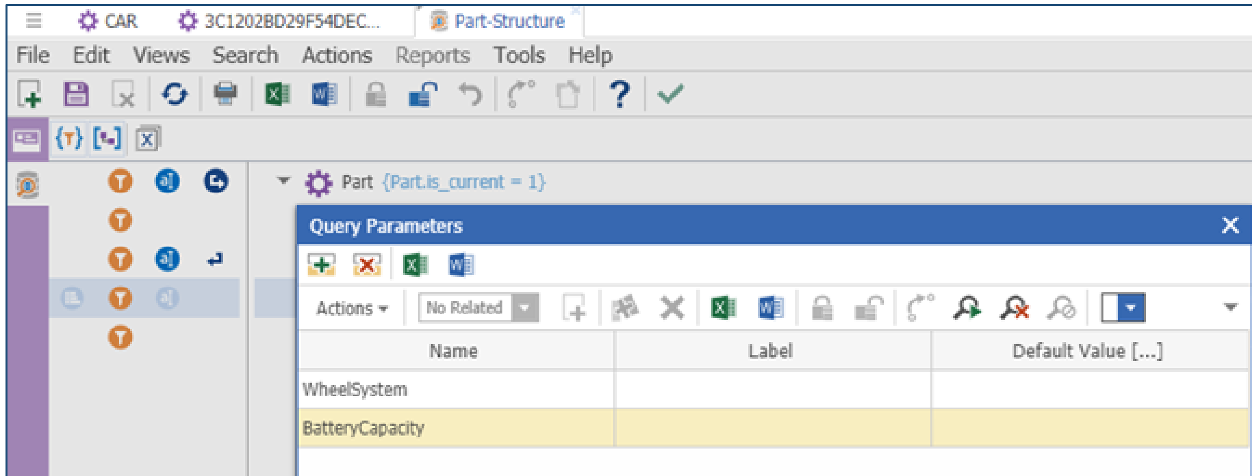


Figure 54.

3. Change the effectivity condition for structure resolution to **BatteryCapacity** variable equals **$BatteryCapacity** parameter.

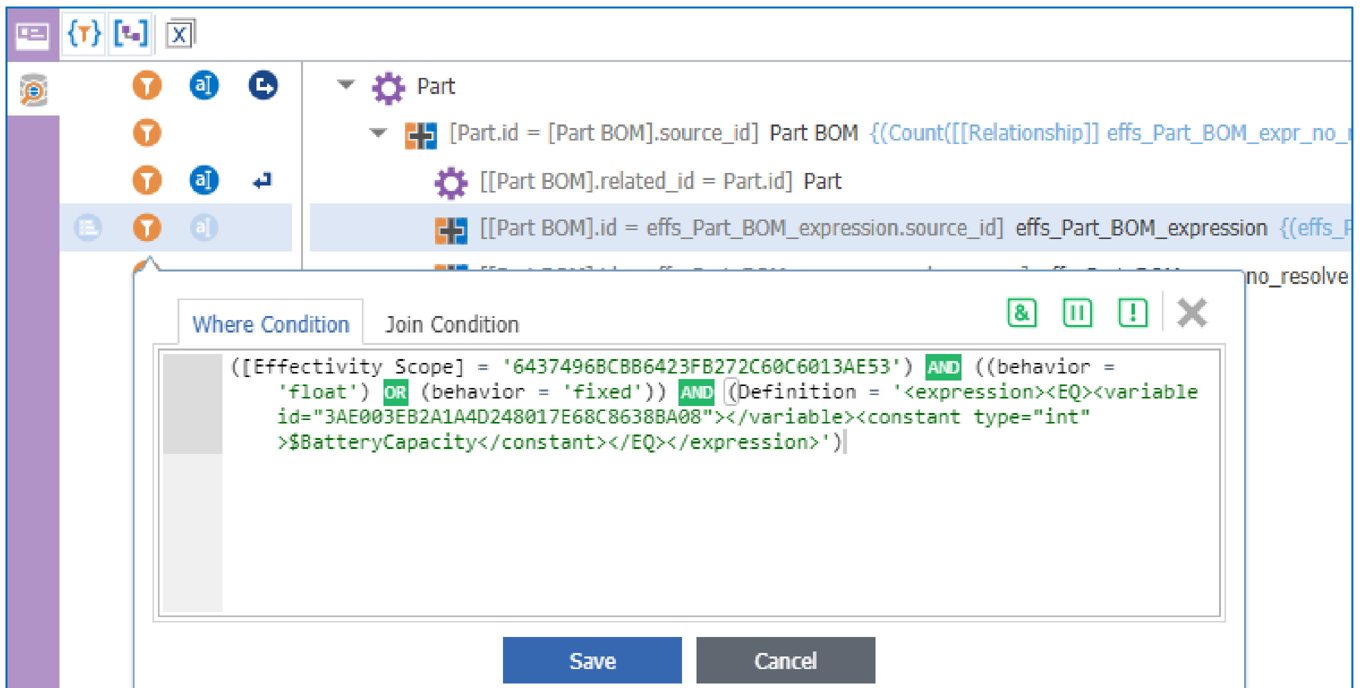


Figure 55.

4. Go to the Tree Grid View item and map the **BatteryCapacity** parameter. Set the **Data Type** value to **Integer**.

©2018 Aras Corporation All Copyrights Reserved.

aras

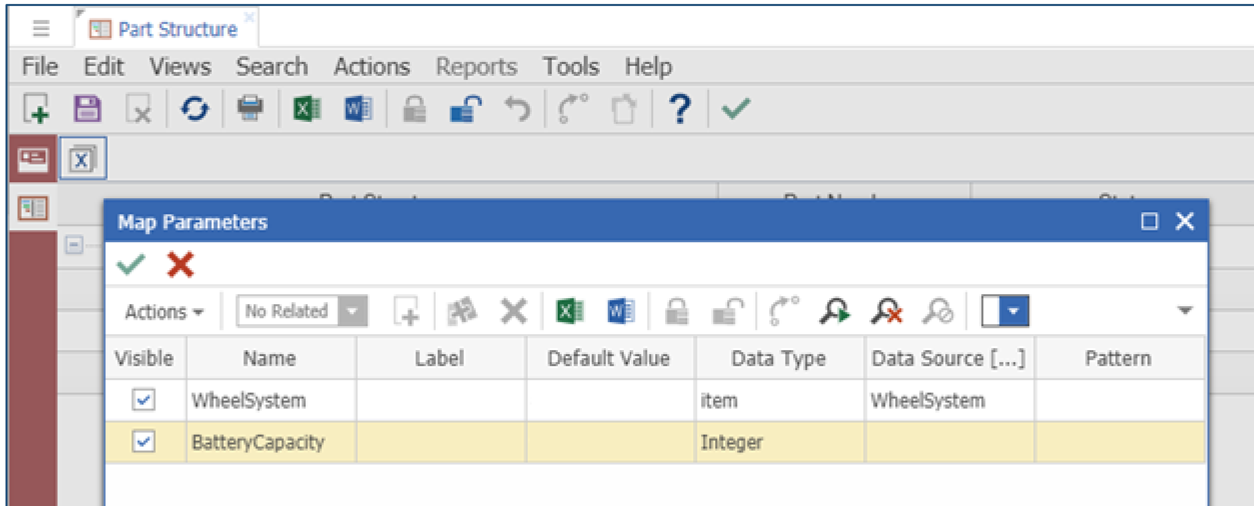


Figure 56.

5. Go to the part item and open the relationship tab in the Tree Grid View. Click the **Modify Parameters** button and change the **BatteryCapacity** parameter value to **111**.

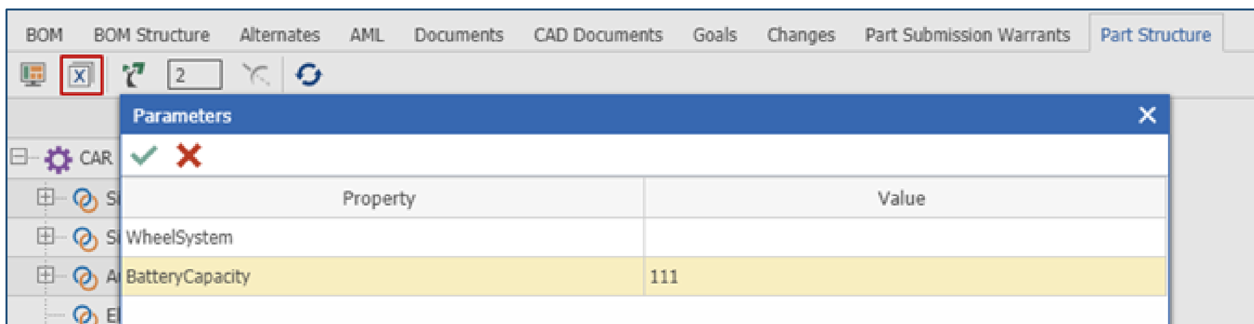


Figure 57.

Note: Due to Tree Grid View parameters functionality, it's required to fill all parameters in the Tree Grid View Parameters dialog for the structure resolution.

©2018 Aras Corporation All Copyrights Reserved.

aras

The "Coupe" related part is displayed because the "111 > 100" condition is fulfilled.

BOM | BOM Structure | Alternates | AML | Documents | CAD Documents | Goals | Changes | Part Submission Warrants | Part Structure

2

Part Structure	Part Number	State
CAR	CAR	Preliminary
Simple Engine 3.0	Simple Engine 3.0	Preliminary
Simple Engine 2.0	Simple Engine 2.0	Preliminary
Automatic GearBox	Automatic GearBox	Preliminary
Coupe	Coupe	Preliminary
Electric Engine 100A	Electric Engine 100A	Preliminary
Electric Engine 50A	Electric Engine 50A	Preliminary
Hybrid Engine	Hybrid Engine	Preliminary
Manual GearBox	Manual GearBox	Preliminary
Sedan	Sedan	Preliminary
Simple Engine 1.8	Simple Engine 1.8	Preliminary

Figure 58.

6. Click the **Modify Parameters** button and change the **BatteryCapacity** parameter value to **90**.

   The "Coupe" related part is not displayed because the BatteryCapacity parameter value 90 conflicts with the "BatteryCapacity >= 100" expression set on the "Part BOM" item.

BOM | BOM Structure | Alternates | AML | Documents | CAD Documents | Goals | Changes | Part Submission Warrants | Part Structure

2

Part Structure	Part Number	State
CAR	CAR	Preliminary
Simple Engine 3.0	Simple Engine 3.0	Preliminary
Simple Engine 2.0	Simple Engine 2.0	Preliminary
Automatic GearBox	Automatic GearBox	Preliminary
Electric Engine 100A	Electric Engine 100A	Preliminary
Electric Engine 50A	Electric Engine 50A	Preliminary
Hybrid Engine	Hybrid Engine	Preliminary
Manual GearBox	Manual GearBox	Preliminary
Sedan	Sedan	Preliminary
Simple Engine 1.8	Simple Engine 1.8	Preliminary

Figure 59.

©2018 Aras Corporation All Copyrights Reserved.

aras

9.3.2.3 Using Tree Grid View Parameters with Date variables

This example uses the Query Builder and Tree Grid View parameters with the "AvailableDate" datetime variable.

1. Create the effectivity expression **AvailableDate >= 2018-03-31** on the "Part BOM" item using "Automatic GearBox" as the related part.

Note: Effectivity Services only supports the "short_date" pattern for DateTime variables in the XML expression. This means that the XML expression must have the datetime value specified using one of the following formats – "yyyy-MM-dd" or "yyyy-MM-ddT00:00:00".

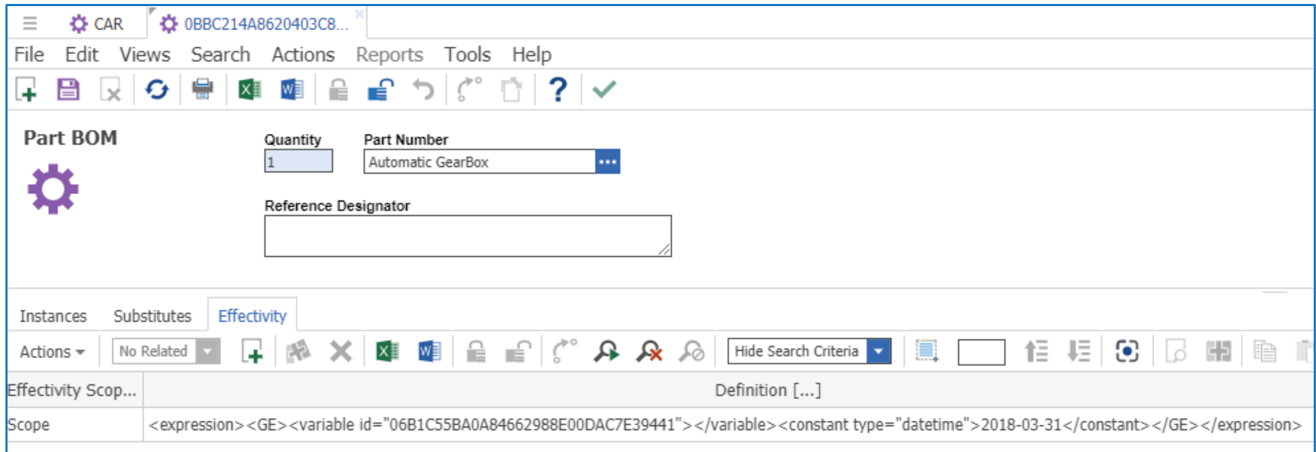


Figure 60.

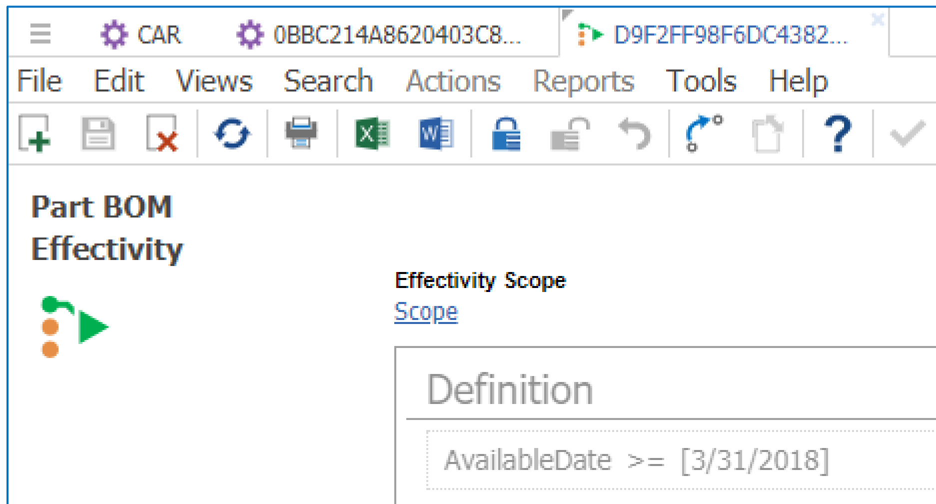


Figure 61.

2. Add the new query parameter **AvailableDate** to the query definition item. If a default value is needed, specify the default value in one of the supported datetime formats - "yyyy-MM-dd" or "yyyy-MM-ddT00:00:00".

©2018 Aras Corporation All Copyrights Reserved.

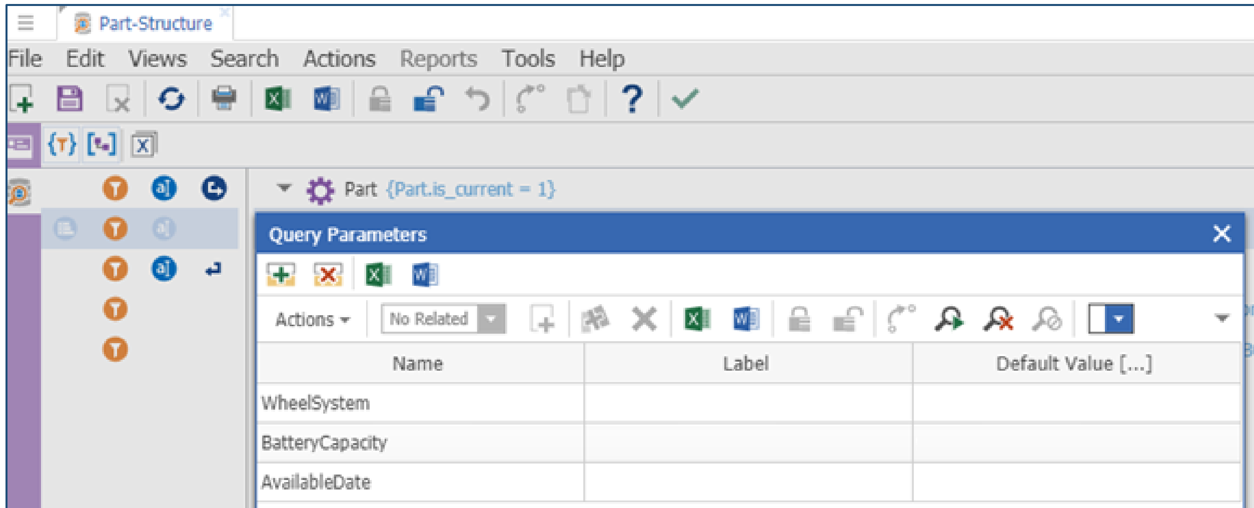


Figure 62.

3. Change the effectivity condition for structure resolution to the **AvailableDate** variable equals **$AvailableDate** parameter.

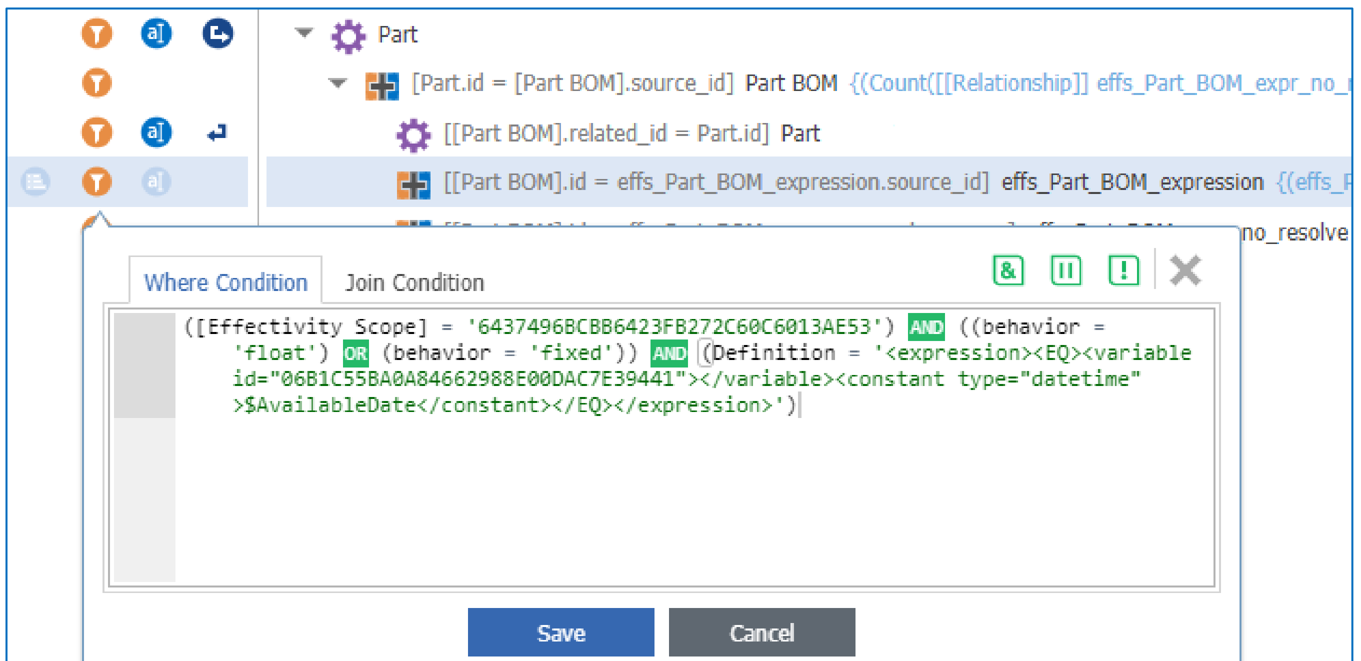


Figure 63.

4. Go to the Tree Grid View item and map the **AvailableDate** parameter. Set the **Data Type** value to "date" and the **Pattern** value to "short_date."

©2018 Aras Corporation All Copyrights Reserved.

aras

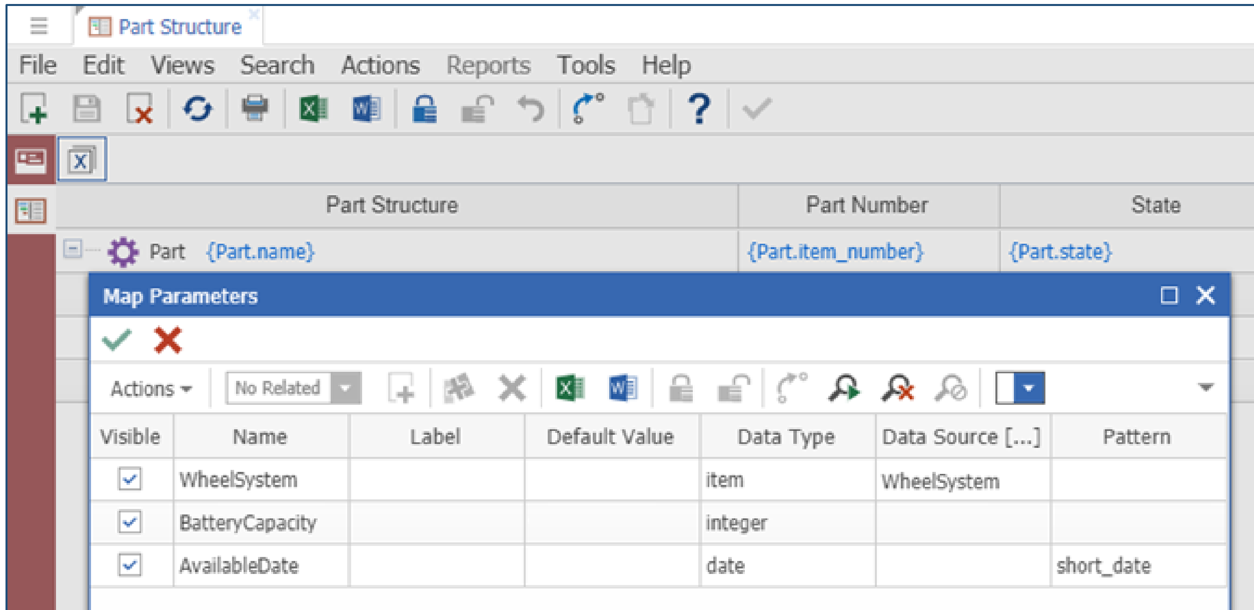


Figure 64.

5. Go to the Part item and open the relationship tab in Tree Grid View. Click the **Modify Parameters** button and change the **AvailableDate** parameter value to 2018-03-31.

Note: In the user interface datetime value appears in the local format.

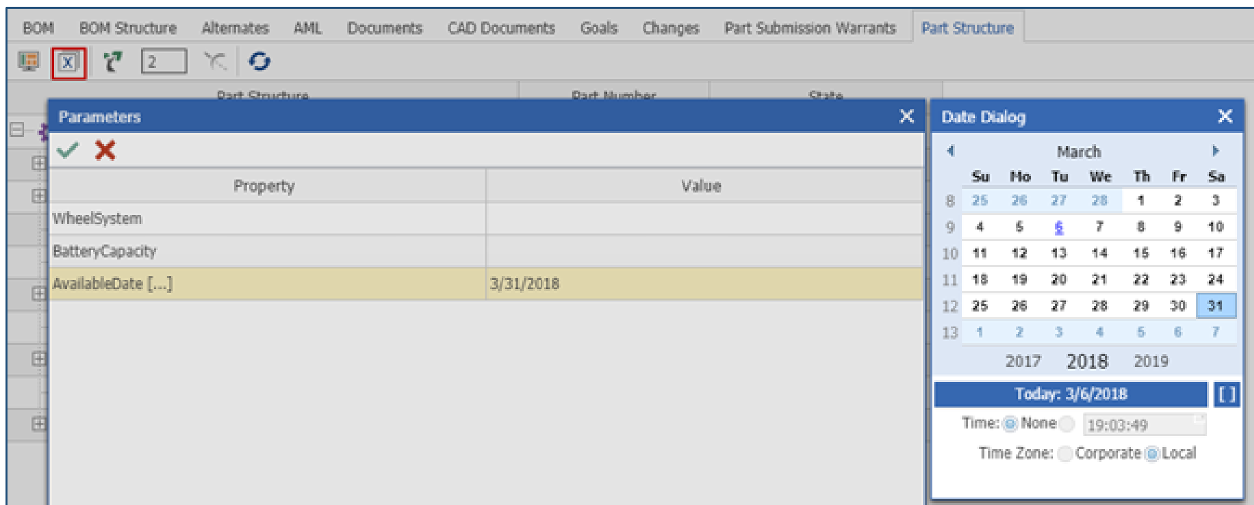


Figure 65.

The "Automatic GearBox" related part is displayed because the "AvailableDate >= 2018-03-31" condition is fulfilled.

©2018 Aras Corporation All Copyrights Reserved.

BOM BOM Structure Alternates AML Documents CAD Documents Goals Changes Part Submission Warrants Part Structure

2

Part Structure	Part Number	State
CAR	CAR	Preliminary
Simple Engine 3.0	Simple Engine 3.0	Preliminary
Simple Engine 2.0	Simple Engine 2.0	Preliminary
Automatic GearBox	Automatic GearBox	Preliminary
Coupe	Coupe	Preliminary
Electric Engine 100A	Electric Engine 100A	Preliminary
Electric Engine 50A	Electric Engine 50A	Preliminary
Hybrid Engine	Hybrid Engine	Preliminary
Manual GearBox	Manual GearBox	Preliminary
Sedan	Sedan	Preliminary
Simple Engine 1.8	Simple Engine 1.8	Preliminary

Figure 66.

6. Click the **Modify Parameters** button and change the **AvailableDate** parameter value to 2018-03-15.
7. The "Automatic GearBox" related part is not displayed because the AvailableDate parameter value 2018-03-15 conflicts with the expression "AvailableDate >= 2018-03-31" set on the "Part BOM" item.

BOM BOM Structure Alternates AML Documents CAD Documents Goals Changes Part Submission Warrants Part Structure

2

Part Structure	Part Number	State
CAR	CAR	Preliminary
Simple Engine 3.0	Simple Engine 3.0	Preliminary
Simple Engine 2.0	Simple Engine 2.0	Preliminary
Coupe	Coupe	Preliminary
Electric Engine 100A	Electric Engine 100A	Preliminary
Electric Engine 50A	Electric Engine 50A	Preliminary
Hybrid Engine	Hybrid Engine	Preliminary
Manual GearBox	Manual GearBox	Preliminary
Sedan	Sedan	Preliminary
Simple Engine 1.8	Simple Engine 1.8	Preliminary

Figure 67.

9.3.2.4 Using Tree Grid View Parameters with String variables

This example uses the Query Builder and Tree Grid View parameters with the "CarModel" string variable.

©2018 Aras Corporation All Copyrights Reserved.

1. Create the effectivity expression **CarModel = Model12** on the "Part BOM" item using the "Sedan" related part.

Note: Latin characters and numbers are the only valid characters for the string variables.

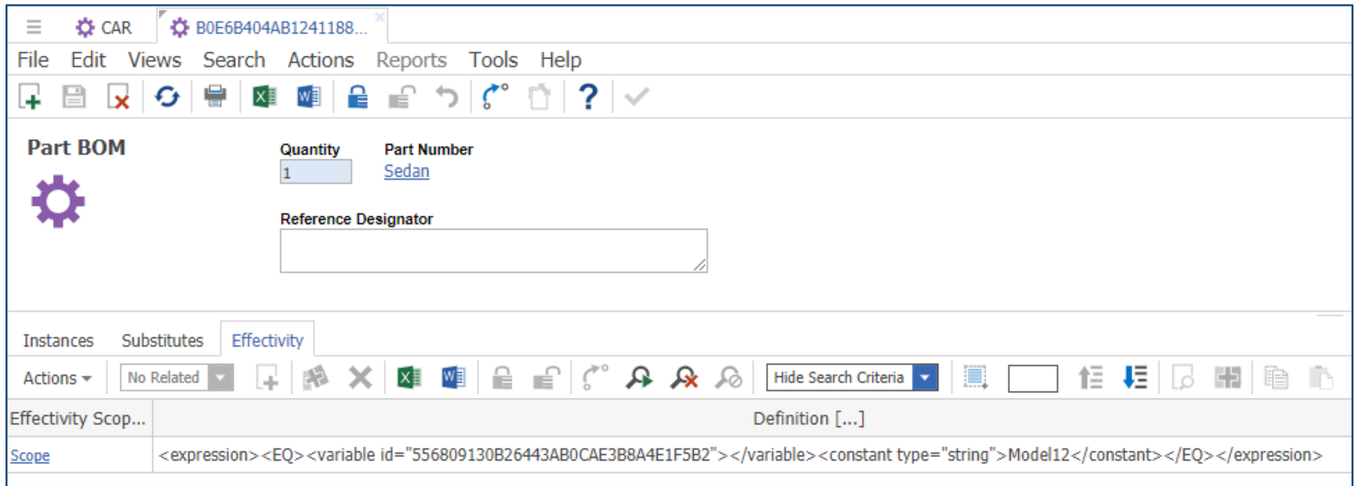


Figure 68.

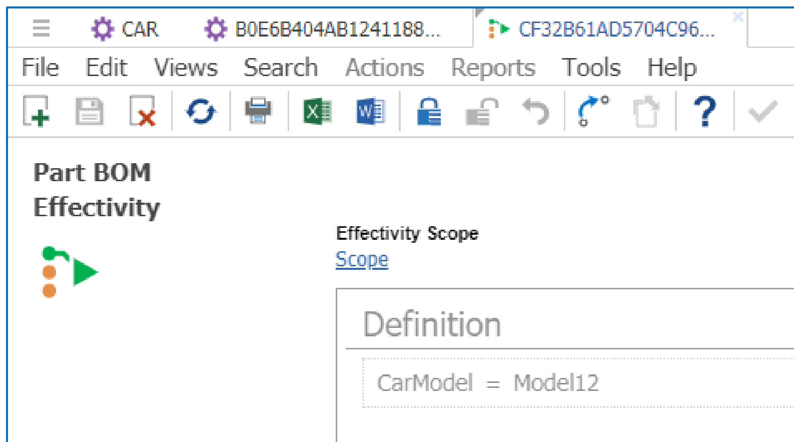


Figure 69.

2. Add the new query parameter **CarModel** to the query definition item.

©2018 Aras Corporation All Copyrights Reserved.

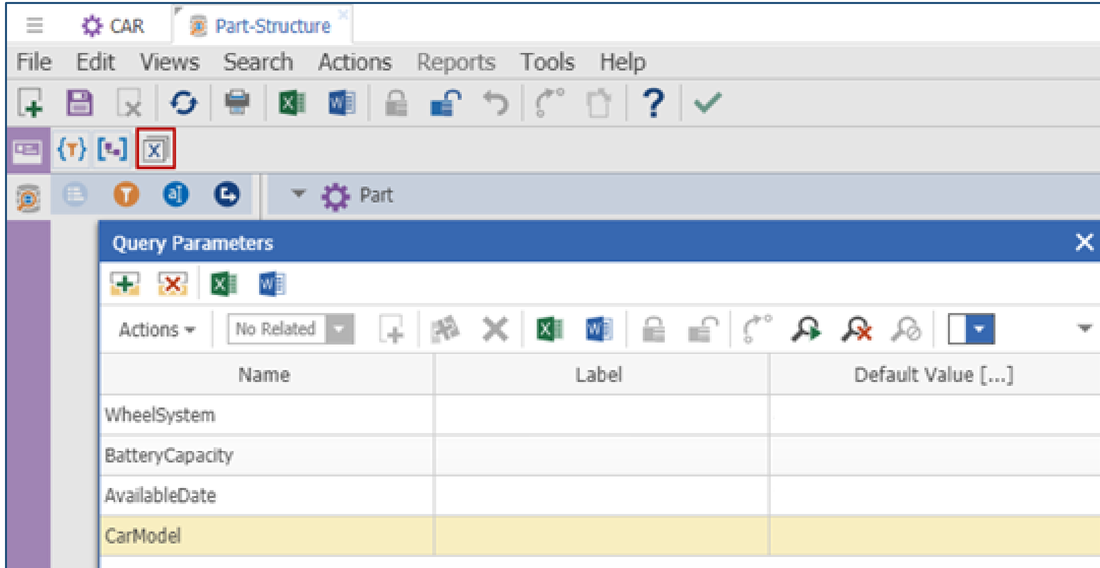


Figure 70.

3. Change the effectivity condition for structure resolution to the **CarModel** variable equals **$CarModel** parameter.

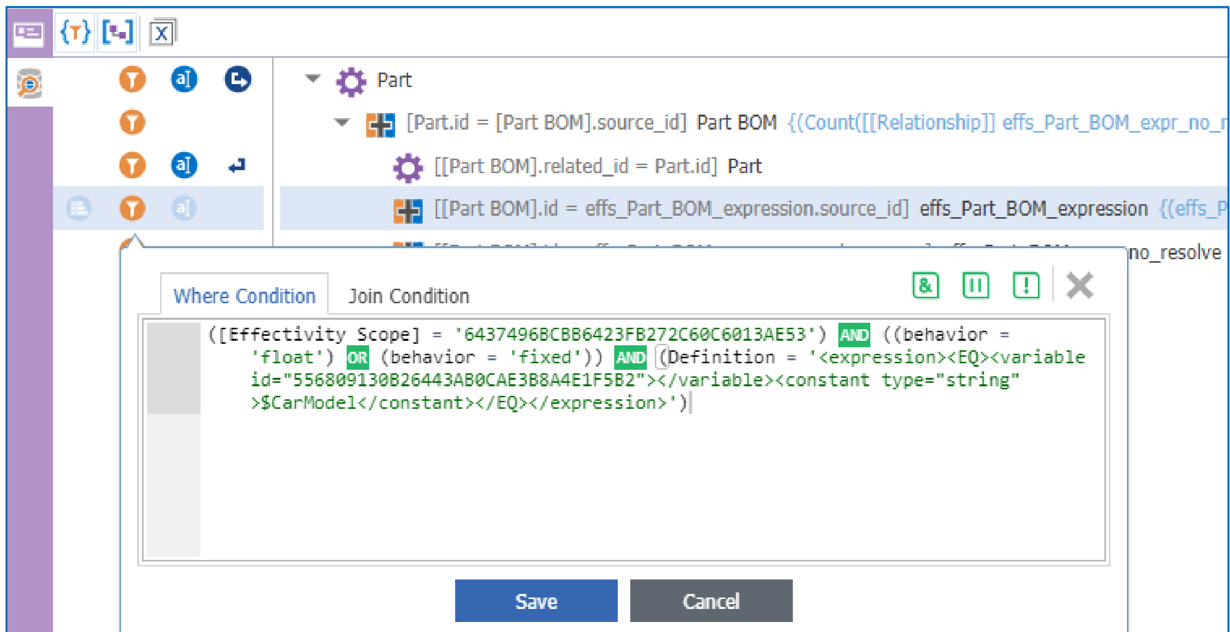


Figure 71.

4. Go to the Tree Grid View item and map the CarModel parameter. Set the **Data Type** value to “string” and the **Pattern** value to “^[0-9A-Za-z]+$” in order to restrict unsupported characters for the query parameter. The Regular expression ^[0-9A-Za-z]+$ enables you to enter only digits and Latin characters in upper or lower case.

©2018 Aras Corporation All Copyrights Reserved.

aras

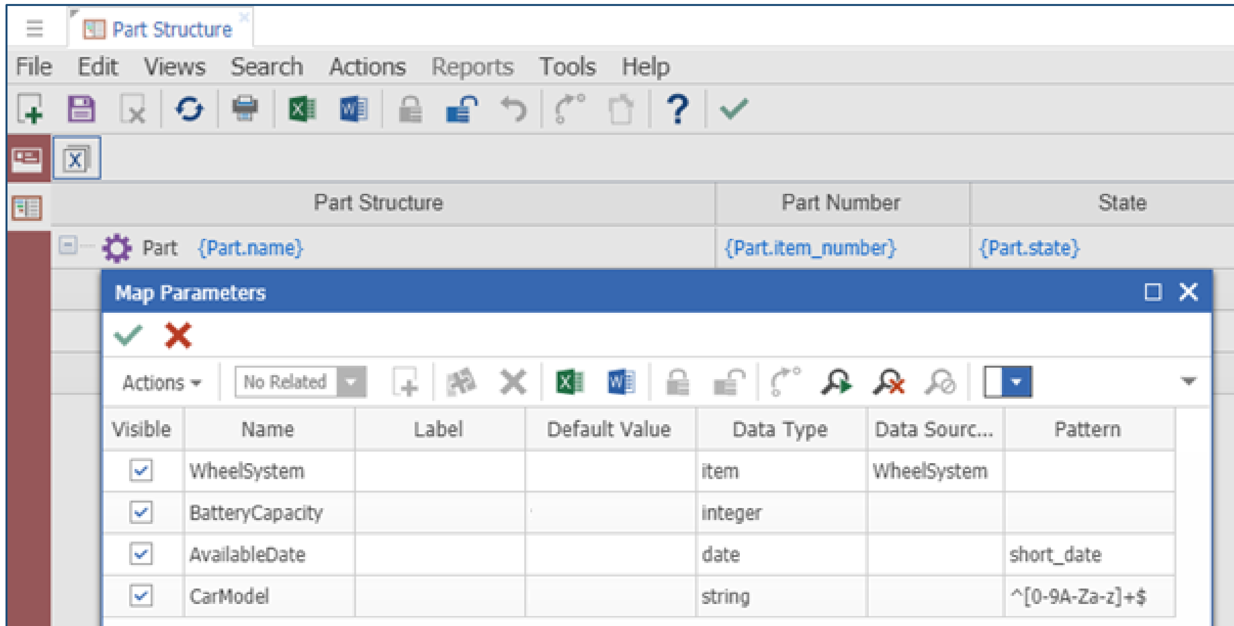


Figure 72.

5. Go to the part item and open the relationship tab in Tree Grid View. Click the **Modify Parameters** button and change the **CarModel** parameter value to **Model12**.

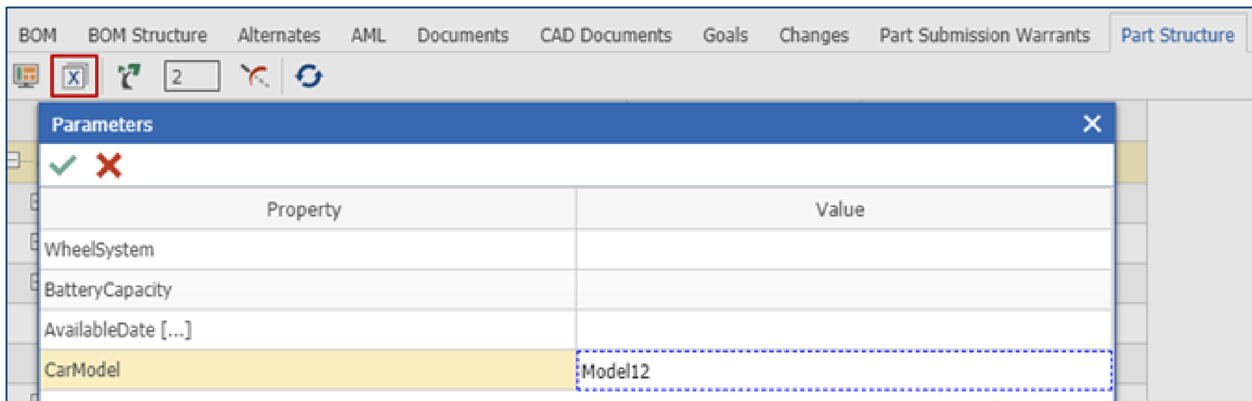


Figure 73.

The “Sedan” related part is displayed because the “CarModel = Model12” condition is fulfilled.

©2018 Aras Corporation All Copyrights Reserved.

BOM | BOM Structure | Alternates | AML | Documents | CAD Documents | Goals | Changes | Part Submission Warrants | Part Structure

Part Structure	Part Number	State
CAR	CAR	Preliminary
Simple Engine 3.0	Simple Engine 3.0	Preliminary
Simple Engine 2.0	Simple Engine 2.0	Preliminary
Automatic GearBox	Automatic GearBox	Preliminary
Coupe	Coupe	Preliminary
Electric Engine 100A	Electric Engine 100A	Preliminary
Electric Engine 50A	Electric Engine 50A	Preliminary
Hybrid Engine	Hybrid Engine	Preliminary
Manual GearBox	Manual GearBox	Preliminary
Sedan	Sedan	Preliminary
Simple Engine 1.8	Simple Engine 1.8	Preliminary

Figure 74.

6. Click the **Modify Parameters** button and change the **CarModel** parameter value to **Model9**.
7. The “Sedan” related part is not displayed because the CarModel parameter value “Model9” conflicts with the expression “CarModel = Model12” set on the “Part BOM” item.

BOM | BOM Structure | Alternates | AML | Documents | CAD Documents | Goals | Changes | Part Submission Warrants | Part Structure

Part Structure	Part Number	State
CAR	CAR	Preliminary
Simple Engine 3.0	Simple Engine 3.0	Preliminary
Simple Engine 2.0	Simple Engine 2.0	Preliminary
Automatic GearBox	Automatic GearBox	Preliminary
Coupe	Coupe	Preliminary
Electric Engine 100A	Electric Engine 100A	Preliminary
Electric Engine 50A	Electric Engine 50A	Preliminary
Hybrid Engine	Hybrid Engine	Preliminary
Manual GearBox	Manual GearBox	Preliminary
Simple Engine 1.8	Simple Engine 1.8	Preliminary

Figure 75.

©2018 Aras Corporation All Copyrights Reserved.

10 Dynamic Tree Grid Control API

The Dynamic Tree Grid Control is available in Effectivity Services. This section describes the API. The Dynamic Tree Grid control is inherited from the TreeGrid component and has all of its public methods, fields, and events.

10.1 Constructor

The constructor supports two parameters:

Name	Type	Description
dom	Object	Required parameter. It is a DOM element that will be used as a control container.
settings	Object	Optional parameter. It contains the following global settings for grid initialization: multiSelect – gives you the ability to select multiple rows in the grid. The default value is false. editable – determines whether a grid can be edited or not. The default value is true. sortable – determines whether or not grid columns can be sorted. The default value is true. draggableColumns – determines whether or not grid columns can be moved. The default value is true. resizable – determines wheter or not columns can be resized. The default value is true. If you do not specify the settings parameter, its default value will be used for grid initialization.

10.2 Public fields

DynamicTreeGrid does not have its own public fields. It uses the base TreeGrid component public fields.

10.3 Events

You can add and remove Events using the “on” and “off” public methods.

Each event listener callback accepts a single parameter: an event object of the type “CustomEvent” containing a “detail” property with specific event information.

©2018 Aras Corporation All Copyrights Reserved.

aras

Note: Column and row indexes are zero-based. Row indexing starts from the first row displayed in a grid and ends at the bottom row of the entire grid. It means that rows whose parents are collapsed/invisible are not counted. The column indexing starts from the first column on the left. All invisible rows and columns have an index of -1.

10.3.1 addRow

Event fired after each row is added to the grid when adding rows using the "addRows" or "loadData" methods.

Detail properties

Name	Type	Description
rowId	String	ID of the added row.
parentId	String	ID of the parent row. Null if added row is root.

10.3.2 addRows

Event fired after all rows are added using the "addRows" or "loadData" methods.

Detail properties

Name	Type	Description
rowIds	Array	Added row IDs.
parentId	String	ID of the parent row. The ID is Null if the added row is root.

10.3.3 removeRow

Event fired after deleting a row from the grid using the "removeRow" method.

Detail properties

Name	Type	Description
rowIds	Array	IDs of the deleted rows. Array contains ID of the deleted row and IDs of all descendant rows.
parentId	String	ID of the parent row. The ID is Null if the deleted

©2018 Aras Corporation All Copyrights Reserved.

aras

		row is a root row.

10.3.4 removeAllRows

Event fired after deleting all rows from the grid using the "removeAllRows" method.

Detail properties

Name	Type	Description
rowIds	Array	IDs of the deleted rows.

10.3.5 addColumn

Event fired after adding a new column using the "addColumn" or "loadData" methods.

Detail properties

Name	Type	Description
columnName	String	Column name.

10.3.6 removeColumn

Event fired after deleting a column using the "removeColumn" method.

Detail properties

Name	Type	Description
columnName	String	Column name.

©2018 Aras Corporation All Copyrights Reserved.

10.4 Common objects description

This section describes common objects frequently used in the DynamicTreeGrid public methods.

10.4.1 Metadata objects

A Metadata object is an object that contains additional information about a cell that is required for the grid editors and formatters to render the cell properly.

A Metadata object can have the following properties:

- formatter – optional property.

  The formatter name used to display cell content. If you do not specify a “formatter” property, the formatter type is determined by the grid based on the cell value. E.g. if the cell value is a string the ‘text’ formatter will be used, and if the cell value is boolean the ‘boolean’ formatter will be used.

- editor – optional property.

  The name of the editor used to display cell content when the cell is in the edit state. If you do not specify an “editor” property, the editor type will be determined by the grid.

- Other properties that can be used in the formatter and editor functions from the “metadata” parameter.

Example:

```
{
        formatter: ‘select’,
        editor: ‘select’,
        options:  [
              {label: 'Red', value: 'id1'},
              {label: 'Green', value: 'id2'},
              {label: 'Blue', value: 'id3'}
        ]
}
```

10.4.2 Column settings object

The Column settings object contains additional settings applicable to the corresponding column.

If a grid is not editable/resizable/sortable, then you cannot edit/resize/sort columns. However, if the grid is editable/resizable/sortable you can adjust these columns. The Column setting does not change when the corresponding grid setting is changed.

The Column settings object can have the following optional properties:

- **resizable** enables you to decide whether or not to resize a column. The default value is obtained from grid settings when adding a new column.
- **editable** enables you to decide whether or not you want to edit cells in this column. The default value is obtained from grid settings when adding a new column. This setting value will be used as a default value for the cell’s editability setting when adding new rows. You can also use the “setCellEditability” and “getCellEditability” methods to set/get cell editability.

©2018 Aras Corporation All Copyrights Reserved.

- **sortable** enables you to decide whether or not you want to sort by this column. The default value is obtained from the grid settings when adding a new column.
- **visible** enables you to decide if the column should be visible or not. The Default value is `true` when adding a new column.

Example:

```
{
        resizable: false,
        sortable: true,
        editable: false,
        visible: true
}
```

10.4.3 Column object

The Column object is used in the "loadData" and "addColumn" methods. It can contain the following properties:

- name – required property. Column name.
- label – optional property. The column name is used as the label if the "label" property is not set.
- width – optional property. The width is calculated automatically based on the column label if the "width" property is not set.
- metadata – optional property. It contains a metadata object for all cells in this column. Column metadata is used only for cells without their own metadata. The Metadata object is described in section 10.4.1.
- settings – optional property containing settings that are applied to the corresponding column. The Column settings object is described in section 10.4.2.

Example:

```
{
        label: 'Column 1',
        width: 100,
        name: 'property1,
        metadata: {
           formatter: 'boolean'
        },
        settings: {
           resizable: false,
           editable: false
        }
}
```

10.4.4 Row object

The Row object is used in the "loadData" and "addRows" methods. It can contain different properties. If the row object property name matches the column name, then its value is used as a cell value. The Row object property is updated automatically when the corresponding grid cell value is changed.

Example:

©2018 Aras Corporation All Copyrights Reserved.

```
{
   property1: 'value 1',
   property2: 'value 2',
   someOtherProperties: 'additional property'
}
```

10.5 Public methods

10.5.1 loadData

Initializes the grid with specified rows and columns.

If the "columns" parameter is not specified, the columns will be obtained automatically from the "rows" parameter. Each unique property of the row object will represent a column with the same name.

If both the "rows" and "columns" parameters are not specified, the method removes all existing rows and columns.

Note: The grid automatically updates the corresponding properties of the provided row object when the cell value is changed.

Optional Input parameters

Name	Type	Description
rows	Array	An array containing the row objects described in section 10.4.4. If the parameter is not an array, the rows are not added.
columns	Array	An array containing column objects described in section 10.4.3. The Array's element position determines the column order.

Return value

Array – an array that contains the IDs of the added rows.

Example of the "rows" array parameter:

```
[
   {
      property1: 'row 1',
      property2: 'value2',
      someOtherProperties: 'additional property'
   },
   {
      property1: 'row 2'
   }
]
```

Example of the "columns" array parameter:

```
 [
```

©2018 Aras Corporation All Copyrights Reserved.

```
        {
                label: 'Column 1',
                width: 55,
                name: 'property1,
                metadata: {
                   formatter: 'select',
                   editor: 'select',
                   options:  [
                      {label: 'Red', value: 'id1'},
                      {label: 'Green', value: 'id2'},
                      {label: 'Blue', value: 'id3'}
                   ]
                }
        },
        {
           name: 'property2',
           settings: {
              resizable: false,
              sortable: true,
              editable: false
           }
        }
]
```

10.5.2 obtainRowID

obtainRowID is an overridable handler which should return a unique row id for the given row object when adding new rows using the "loadData" or "addRows" methods. It returns null by default. The Unique row id will be generated automatically if the handler returns a falsy value, e.g. false, empty string, null or undefined. If the handler returns a truthy value which is not a string or grid that already contains a row with the same id, an error will be thrown in the "loadData" or "addRows" methods.

Input parameters

Name	Type	Description
rowInfo	Object	Row object which is used to add a new row.
parentId	String	Parent row ID. The ID will be Null if the row is added as root.

Return value

String – row ID.

Example:

```
dynamicTreeGrid.obtainRowId = function(rowObj, parentId) {
   return rowObj.id_property;
};
```

©2018 Aras Corporation All Copyrights Reserved.

10.5.3 addRows

Adds rows to the specified position in the grid.

Input parameters

Name	Type	Description
rows	Array	Required parameter. Arrays containing row objects are described in section 10.4.4. If the parameter is not an array, the rows will not be added.
parentId	String	Optional parameter. ID of the parent row. If the "parentId" parameter is null or undefined, rows will be added as roots.
position	Number	Optional parameter. It is a zero-based index that is used to insert new rows into the child row IDs array of the parent row (or roots array if parentId is not specified). The Index can be greater than or equal to 0 and less than or equal to the children array length. If the "position" parameter is not specified, new child rows will be appended to the end of the children array. The getChildRowIds method can be used to find out the row index in the children array.
doRender	Boolean	Optional parameter. Renders grid after adding rows. The default value is true.

Return value

Array – array which contains IDs of the added rows.

10.5.4 removeRow

Removes the row with the specified ID from the grid and grid rows collection.

Input parameters

Name	Type	Description
rowId	String	Row ID.

©2018 Aras Corporation All Copyrights Reserved.

aras

10.5.5 removeAllRows

Removes all rows from the grid and the grid rows collection.

10.5.6 selectRow

Selects row with the specified ID. If grid's "multiSelect" setting is set to false, the specified row becomes the only selected row. Otherwise the specified row becomes selected and all previously selected rows stay selected.

Input parameters

Name	Type	Description
rowId	String	Row ID.

10.5.7 deselectRow

Cancels the selection of the specified row.

Input parameters

Name	Type	Description
rowId	String	Row id.

10.5.8 addColumn

Adds a column to the grid.

Input parameters

Name	Type	Description
column	Object	Required parameter. The Column object is described in the section 10.4.3.
position	Number	Optional parameter. Zero based column index. If its position is not specified, the column will be appended after existing columns.

©2018 Aras Corporation All Copyrights Reserved.

10.5.9 removeColumn

Removes the specified column from the grid and the grid columns collection.

Input parameters

Name	Type	Description
name	String	Column name.

10.5.10 getChildRowIds

Returns an Array containing the child row IDs for the specified parent row. This method returns the IDs of the root rows if the "parentId" parameter is null or undefined.

Input parameters

Name	Type	Description
parentId	String	Optional parameter. ID of the parent row.

Return value

Array – array with child row IDs or an empty array if there are no children.

10.5.11 getRowUserData

Gets user data stored by the specified row and key.

Input parameters

Name	Type	Description
rowId	String	Row ID.
key	String	Key for storing user data.

Return value

Anything - User data stored by the specified row and key.

Example:

```
let itemtype = grid.getRowUserData(rowId, 'itemtype');
```

©2018 Aras Corporation All Copyrights Reserved.

aras

10.5.12 setRowUserData

Sets user data for the specified row and key.

Input parameters

Name	Type	Description
rowId	String	Row Id
key	String	Key for storing user data
value	Anything	Value

Example:

```
grid.setRowUserData(rowId, 'itemtype', 'Part');
```

10.5.13 setCellEditability

Sets cell edit availability.

Input parameters

Name	Type	Description
rowId	String	Required parameter. Row ID.
columnName	String	Required parameter. Column name.
editable	Boolean	Optional parameter. Can a cell be edited or not. Default value is true.

©2018 Aras Corporation All Copyrights Reserved.

10.5.14 getCellEditability

Returns `true` if the specified cell is editable, otherwise it returns `false`.

Input parameters

Name	Type	Description
rowId	String	Row Id.
columnName	String	Column name.

Return value

Boolean – is cell editable or not.

10.5.15 setCellValue

Sets cell value and updates the corresponding property in the user row object.

Input parameters

Name	Type	Description
rowId	String	Row ID
columnName	String	Column name
value	Anything	Cell value

10.5.16 getCellValue

Gets the cell value.

Input parameters

Name	Type	Description
rowId	String	Row ID
columnName	String	Column name

Return value

Anything – cell value.

©2018 Aras Corporation All Copyrights Reserved.

10.5.17 getColumnCount

Gets a count of all columns (visible and invisible).

Return value

Number

10.5.18 getVisibleColumnCount

Gets a count of visible columns.

Return value

Number

10.5.19 getColumnIndex

Gets a column index by the column name. Returns -1 if column is not found or invisible.

Input parameters

Name	Type	Description
columnName	String	Column name.

Return value

Number – column index

10.5.20 getColumnName

Gets the column name using the column index.

Input parameters

Name	Type	Description
columnIndex	Number	Column index

Return value

String – column name

10.5.21 getColumnOrder

Gets Array with the names of visible columns in the same order as they are displayed in the grid.

Return value

Array

©2018 Aras Corporation All Copyrights Reserved.

aras

10.5.22 getAllColumnNames

Gets Array with the names of all columns in the order they are added to the grid.

Return value

Array

10.5.23 getParentId

Gets the ID of the parent row for the specified row. Returns null if the specified row is root and has no parent.

Input parameters

Name	Type	Description
rowId	String	Row Id.

Return value

String – parent row ID.

10.5.24 containsRow

Returns `true` if the grid has a row with the specified ID in the rows collection, otherwise it returns `false`.

Input parameters

Name	Type	Description
rowId	String	Row Id.

Return value

Boolean

10.5.25 containsColumn

Returns `true` if the grid has a column with the specified name in the columns collection, otherwise it returns `false`.

Input parameters

Name	Type	Description
columnName	String	Column name.

Return value

Boolean

©2018 Aras Corporation All Copyrights Reserved.

10.5.26 expandAll

Expands all rows with their descendants.

10.5.27 collapseAll

Collapses all rows with their descendants.

10.5.28 isRowExpanded

Returns `true` if the specified row is expanded, otherwise it returns `false`.

Input parameters

Name	Type	Description
rowId	String	Row Id.

Return value

Boolean

10.5.29 isRowVisible

Returns `true` if the specified row is visible, otherwise it returns `false`.

Input parameters

Name	Type	Description
rowId	String	Row Id.

Return value

Boolean

10.5.30 getVisibleRowCount

Gets the number of visible rows that can be displayed in the grid.

Note: The number of visible rows is not the total rows count. For example, if the grid has only one root row with 10 children and the root row is collapsed, the getVisibleRowCount() method will return 1.

Return value

Number

10.5.31 getAllRowCount

Gets the number of all rows (visible and invisible) contained in the rows collection.

Return value

©2018 Aras Corporation All Copyrights Reserved.

Number

10.5.32 getVisibleRowIds

Gets the Array with the IDs of visible rows displayed in the grid.

Return value

Array

10.5.33 getAllRowIds

Gets the Array with the IDs of all rows contained in the rows collection.

Return value

Array

10.5.34 getRowId

Gets the row ID by the row index.

Input parameters

Name	Type	Description
rowIndex	Number	Row index (zero based, from "top" to "bottom").

Return value

String – row ID.

10.5.35 getRowIndex

Gets the row index using the row ID. The method returns -1 if the row is not found or not visible.

Input parameters

Name	Type	Description
rowId	String	Row ID

Return value

Number – row index.

©2018 Aras Corporation All Copyrights Reserved.

10.5.36 moveColumn

Moves a column to the specified position.

Input parameters

Name	Type	Description
columnName	String	Column name
columnIndex	Number	New column position – zero based index.

10.5.37 setColumnSettings

Sets settings for the specified column.

Input parameters

Name	Type	Description
columnName	String	Column name
settings	Object	The Column settings object described in the section 10.4.2 **Note:** If a column setting is not specified in the column object, its value will not be changed.

10.5.38 getColumnSettings

Gets settings for the specified column.

Input parameters

Name	Type	Description
columnName	String	Column name.

Return value

Object - column settings object described in section 10.4.2.

10.5.39 setColumnMetadata

Sets metadata for the specified column that will be used by the column cells without their own metadata.

©2018 Aras Corporation All Copyrights Reserved.

Input parameters

Name	Type	Description
columnName	String	Column name
metadata	Object	Metadata object described in the section 10.4.1.

10.5.40 getColumnMetadata

Gets metadata for the specified column. Returns null if the column has no metadata.

Input parameters

Name	Type	Description
columnName	String	Column name

Return value

Object – metadata object described in section 10.4.1.

10.5.41 setColumnVisibility

Sets column visibility.

Input parameters

Name	Type	Description
columnName	String	Required parameter. Column name.
visible	Boolean	Optional parameter. Column visibility. The default value is true.

10.5.42 getColumnVisibility

Returns `true` if the specified column is visible, otherwise it returns `false`.

Input parameters

Name	Type	Description
columnName	String	Column name

©2018 Aras Corporation All Copyrights Reserved.

Return value

Boolean

10.5.43 setColumnLabel

Sets the label for the specified column.

Input parameters

Name	Type	Description
columnName	String	Column name
label	String	Column label.

10.5.44 getColumnLabel

Gets the label for the specified column.

Input parameters

Name	Type	Description
columnName	String	Column name

Return value

String - column label.

10.5.45 setColumnWidth

Sets the width for the specified column.

Input parameters

Name	Type	Description
columnName	String	Column name
width	Number	Column width

©2018 Aras Corporation All Copyrights Reserved.

10.5.46 getColumnWidth

Gets the width for the specified column.

Input parameters

Name	Type	Description
columnName	String	Column name

Return value

Number - column width.

10.5.47 setCellMetadata

Sets metadata for the specified cell.

Input parameters

Name	Type	Description
rowId	String	Row Id
columnName	String	Column name
metadata	Object	Metadata object described in the section 10.4.1

10.5.48 getCellMetadataOnly

Gets metadata for the specified cell. Returns null if the cell has no metadata (even if the column has metadata).

Input parameters

Name	Type	Description
rowId	String	Row Id
columnName	String	Column name

Return value

Object – metadata object described in section 10.4.1.

©2018 Aras Corporation All Copyrights Reserved.

aras