xProperties in AML

This section describes how to define attributes and xProperties programmatically in AML.

Using the @set attribute

You can perform the following operations on an xProperty when you are updating or adding an Item:

- Set a value.
- Explicitly define it
- Change the private permissions

You must add the @set attribute to an xProperty node in order to specify the operation to be performed. The following is the list of valid values associated with the @set attribute:

- “value”
- “explicit”
- “permission_id”
- Any combination of the values listed here, using the “|” as the divider (for example, “explicit|value” means “perform both of these operations”).

If you do not include the @set attribute, update operations will not occur.

Explicitly Defining an xProperty on an Item

The difference between xProperties and standard properties is that you have to define an xProperty for an item before you can perform an operation (for example get/set value) with it.

You can either define the xProperty explicitly or implicitly. To explicitly define an xProperty you have to add the @explicit and @set attributes to the Property node. If you do not specify the @set attribute or if it does not contain the value “explicit”, the “@explicit” attribute is ignored.

The following example shows how to add an item and define an xProperty programmatically. The default value is set for the “xp-cost” property because the “set” attribute does not contain a “value” string:

```
<Item type="Part" action="add">
<xp-cost set="explicit" explicit="1"/>
<cost>128</cost>
</Item>
```

The following example updates an item and defines the xProperty:

```
<Item type="Part" action="update" id="0FA8ED...">
<xp-cost set="explicit" explicit="1"/>
<cost>128</cost>
</Item>
```

This example updates the item, defines an xProperty and sets its value:

```
<Item type="Part" action="update" id="0FA8ED...">
<xp-cost set="value|explicit" explicit="1">100<xp-cost/>
<cost>128</cost>
</Item>
```

This example updates the item and sets the value for an already defined xProperty:

```
<Item type="Part" action="update" id="0FA8ED...">
<xp-cost set="value">100<xp-cost/>
<cost>128</cost>
</Item>
```

This is an example of making an explicitly defined xProperty undefined:

```
<Item type="Part" action="update" id="0FA8ED...">
<xp-cost set="explicit" explicit="0"/>
</Item>
```

Resolving Ambiguous Property Names

Use the “xp-“ prefix to distinguish an xProperty from a standard property in AML, For example, the xProperty name for the cost property would be xp-cost. This technique guarantees that names are unique because standard properties cannot use a hyphen as part of a name.

Important

You cannot use Namespaces resolve ambiguous property names because they are used in AML to return multilingual strings. It does not work for an xProperty with a multilingual String data type. Refer to the following example. The following AML is invalid.

```
<Item xmlns:xProp="http://www.aras.com/xProperties">
<mls>String</mls>
<i18n:mls xml:lang="ru">Строка</i18n:mls>
<xProp:mls>Other String</xProp:mls>
<i18n:xPropr:mls xml:lang="ru">Другая Строка</i18n:xPropr:mls>
</Item>
```

Assigning an xPropertyDefinition to an ItemType

When you assign an xProperty to an ItemType, it is referenced in AML as a property of the ItemType. If you do not define an xProperty on an item, it is ignored. The following example gets all the items from the database where weight is equal to 15 and the xProperty cost is equal to 10:

```
<Item type="Part" action="get" select="item_number, cost, xp-cost">
<xp-cost>10</xp-cost>
<weight>15</weight>
</Item>
<Result>
<Item type="Part" typeId="3B135425…" id="0FA8ED…">
<item_number>999-888</item_number>
<cost>127</cost>
<xp-cost>10</xp-cost>
</Item>
</Result>
```

Get items example if the user doesn’t have permissions on the xProperty:

Get all part items from the database, where the weight is equal to 15 and the xProperty cost is equal to 10. For each found item return the item_number, cost and xProperty cost (defined explicitly). User does NOT have “can get” permissions on the xProperty cost.

```
<Item type="Part" action="get" select="item_number, cost, xp-cost">
<xp-cost>10</xp-cost>
<weight>15</weight>
</Item>
<Result>
<Item type="Part" typeId="3B135425…" id="0FA8ED…">
<item_number>999-888</item_number>
<cost>127</cost>
<xp-cost is_null="0"></xp-cost>
</Item>
</Result>
```

Get items with “select=*” example:

Get all part items from the database, where the weight is equal to 15 and the xProperty cost (defined explicitly) is equal to 10. The @select attribute is not specified. The xProperty length is defined on the resulting item and has a NULL value. For each found item return all standard properties and all defined xProperties. All properties with the value NULL are not returned.

```
<Item type="Part" action="get" select="*">
<xp-cost>10</xp-cost>
<weight>15</weight>
</Item>
<Result>
<Item type="Part" typeId="3B135425…" id="0FA8ED…">
<item_number>999-888</item_number>
<cost>127</cost>
... all NOT NULL standard properties here
<xp-cost>10</xp-cost>
... all NOT NULL xProperties defined on this PART item here
</Item>
</Result>
```

Get items without “select” :

Get all part items from the database, where the weight is equal to 15 and the xProperty cost is equal to 10. The xProperty length is defined on the resulting item implicitly. For each found item return all standard properties and all defined xProperties.

```
<Item type="Part" action="get">
<xp-cost>10</xp-cost>
<weight>15</weight>
</Item>
<Result>
<Item type="Part" typeId="3B135425…" id="0FA8ED…">
<item_number>999-888</item_number>
<cost>127</cost>
<team_id is_null="1"/>
... all standard properties here
<xp-cost>10</xp-cost>
<xp-length is_null="1"/>
... all xProperties defined on this PART item here
</Item>
</Result>
```

Update item example:

Update the standard property “Cost” value to 128 and xProperty “Cost” (defined explicitly or implicitly) value to 100:

```
<Item type="Part" action="update" id="0FA8ED...">
<xp-cost set="value">100</xp-cost>
<cost>128</cost>
</Item>
```

Using @defined_as to Filter xProperties

The following conditions enable you to filer items according to their defined status:

- “is defined”
- “is not defined”

Use the @defined_as attribute to filter items according to how they are defined. You can use the following values with the attribute:

- “class” indicates that the xProperty is defined using classification. It does not matter whether or not the xProperty is defined explicitly.
- “explicit” indicates that the xProperty is defined explicitly. It does not matter if the xProperty is defined/not defined explicitly).
- “class|explicit” indicates that the xProperty is defined both by classification and explicitly.

The following example gets all the Parts with a defined xProperty from the database. It does not matter how the xProperty is defined:

```
<AML>
<Item type="Part" action="get">
<xp-cost condition="is defined"/>
</Item>
</AML>
```

The following example gets all Parts from the database that have an explicitly-defined xProperty:

```
<AML>
<Item type="Part" action="get">
<xp-cost condition="is defined" defined_as="explicit"/>
</Item>
</AML>
```

The following example gets all Parts from the database, which have an xProperty that is NOT defined using classification:

```
<AML>
<Item type="Part" action="get">
<xp-cost condition="is not defined" defined_as="class"/>
</Item>
</AML>
```

Getting Additional xProperty Information

Every xProperty has a complex structure that includes the following information:

- A value
- A definition
- Private permission
- Flags

> **Important**
>
> Due to performance considerations, the default server only returns the xProperty value in the AML response.

You can extend the syntax of the @select attribute to get additional information about the xProperties defined for an item by using the following attributes:

- $value
- @permission_id
- @explicit
- @defined_as

The following example gets all Parts contained in the database. It includes the item_number and all the defined xProperties for each returned item in the response:

```
<AML>
<Item type=Part" action="get" select="item_number, xp-*(@defined_as)"></Item>
</AML>
<Result>
<Item type="Part" typeId="3B135425…" id="0FA8ED…">
<item_number>999-888</item_number>
<xp-length defined_as="class" is_null="0"/>
</Item>
<Item type="Part" typeId="4B135425…" id="1FA8ED…">
<item_number>999-777</item_number>
<xp-length defined_as="class" is_null="0"/>
<xp-width defined_as="class|explicit" is_null="0"/>
<xp-height defined_as="explicit" is_null="0"/>
</Item>
</Result>
```

The following example gets additional information about the defined xProperty (permission_id, explicit):

```
<Item type="Part" action="get" select="item_number, xp-cost($value, @permission_id, @explicit), xp-length
</Item>
<Result>
<Item type="Part" typeId="3B135425…" id="0FA8ED…">
<item_number>999-888</item_number>
<xp-cost explicit="1">10</xp-cost>
<xp-length permission_id="123..." explicit="0" />10<xp-length>
</Item>
</Result>
```

The following example gets additional information about the defined xProperty (is_defined):

```
<Item type="Part" action="get"
select="item_number, xp-*($value, @defined_as)">
</Item>
<Result>
<Item type="Part" typeId="3B135425…" id="0FA8ED…">
<item_number>999-888</item_number>
<xp-height defined_as="class|explicit"/>10<xp-height>
<xp-cost defined_as="explicit">10</xp-cost>
<xp-length defined_as="class"/>10<xp-length>
</Item>
</Result>
```

Changing Private Permissions

You must add the @permission_id and @set attributes to a property node to be able to set private permissions for an xProperty. The @set attribute must contain the permission_id value. The following

example sets private permissions without changing the xProperty value:

```
<Item type="Part" action="update" id="0FA8ED...">
<xp-cost set="permission_id" permission_id="1FBA8E6…"></xp-cost>
</Item>
```

The following example sets private permissions and changes the xProperty value:

```
<Item type="Part" action="update" id="0FA8ED...">
<xp-cost set="permission_id|value" permission_id="1FBA8E6…">100</xp-cost>
</Item>
```

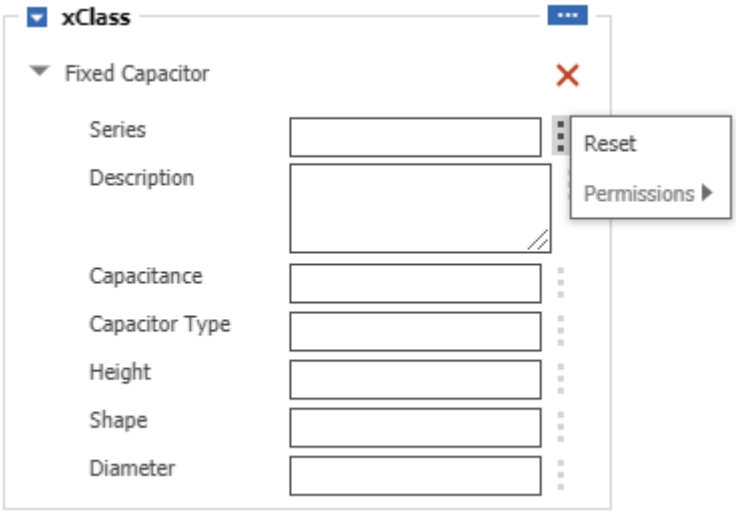
The following example sets private permissions to NULL value:

```
<Item type="Part" action="update" id="0FA8ED...">
<xp-cost set="permission_id" permission_id=""></xp-cost>
</Item>
```

The following example specifies that the permission should not be changed without using the @set attribute:

```
<Item type="Part" action="update" id="0FA8ED...">
<xp-cost permission_id=""></xp-cost>
</Item>
```

This AML does NOT change private permissions (because @set attribute is not added) and will NOT set value to NULL (because the absense of the @set attribute is the equivalent of “set=’’”). You can use the UI to change a Private Permission. In the Item Form, for the given xProperty click the button next the xProperty field to make the change:



Querying xProperties across Item Types

Use the xPropertyContainerItem polyitem type to retrieve values for multiple item types. Aras Innovator automatically adds the xPropertyContainerItem poly source to each item type that contains at least one allowed xProperty. The list of allowed xProperties for xPropertyContainerItem is the union of all allowed xProperties from the xPropertyContainerItem poly sources, as shown in the following example:

```
<Item type="xPropertyContainerItem" action="get" select="xp-length, xp-cost, item_id, item_type_id">
<xp-cost>10</xp-cost>
<xp-weight>15</xp-weight>
</Item>
```

You can also search for xProperties across ItemTypes by selecting **My Innovator Extended Properties Search** in the TOC. A list of ItemTypes associated with xProperties appears in the Main Grid.

Filtering Items by Item Type Name

You can use the xPropertyContainerItem ItemType property to filter items by item type. The @condition and @id attributes enable you to filter items by their type names, as shown in the following example:

```
<AML>
<Item type="xPropertyContainerItem" action="get">
<xp-cost>100</xp-cost>
<itemtype condition="in" by="id">
<Item type="ItemType" select="id">
<name condition="ne">Part</name>
</Item>
</itemtype>
</Item>
</AML>
```

The previous AML statement generates the following SQL:

```
SELECT … FROM [xPropertyContainerItem]
WHERE itemtype IN (SELECT id FROM secured.[ItemType] WHERE name <> 'Part')
```

In this case the:

- `condition="in"` attribute tells the server that it is required to filter using a subquery.
- `by="id"` attribute indicates which property has to be used from the subquery for filtering.
- Secured function has to be used in the subquery.

Using the “condition” and “by” attributes to filter items by ANY property of ANY type

You can use the `condition="in" by="…"` attributes not only for the “itemtype” property of a poly item, but for ANY property of ANY item type.For instance, Item Type *A* has a ‘weak’ reference to item type *B* not through *B.id* but through *B.name.* Item type *A* has a property *reference_to_b* which is a value of (unique) name of an item of type *B.* Now I want to use AML to find all items of type *A* that reference items of type *B* which have *cost > 10.* Then I can issue the following AML:

```
<Item type="A" action="get" select="…">
<reference_to _b condition="in" by="name">
<Item type="B" action="get" select="name">
<cost condition="gt">10</cost>
</Item>
</reference_to_b>
</Item>
```

xProperty of Data Type Item

Extended Properties (xProperties) of type Item now support Float behavior and can now be configured to float to the most recent version of the referenced item, providing greater flexibility

where Float behavior is needed.

Using $value, @keyed_name, @type in a select attribute

The following example uses the property *xp-document* of type item where the data_source is Document. The Document ItemType has the properties *name* and *description*. In order to add *name* and *description* to the output of an AML request, you have to use **$value** to get access to the *name* and *description* properties of *xp-document*.

```
Query:
<Item type="Part" select="xp-document($value(name,description))" />
Return:
<Item type="Part" id="%Part.ID%">
<xp-document keyed_name="xp_doc_keyed_name" type="Document">
<Item type="Document" …>
<name>...</name>
<description>...</description>
</Item>
</xp-document>
</Item>
```

If the names of properties of *xp-document* are specified without using **$value**, a null value is returned as shown here:

```
Query:
<Item type="Part" select="xp-document(@defined_as, name, description)" />
Return:
<Item type="Part" id="%Part.ID%">
<xp-document keyed_name="xp_doc_keyed_name" type="Document" defined_as="..." is_null="0"/>
</Item>
```

Other Examples:

```
Query:
<Item type="Part" select="xp-document($value)" />
Return:
<Item type="Part" id="%Part.ID%" …>
<xp-document keyed_name="xp_doc_keyed_name" type="Document">%Document.ID%</xp-document>
</Item>
Query:
<Item type="Part" select="xp-document($value(*))" />
Return:
<Item type="Part" id="%Part.ID%" …>
<xp-document keyed_name="xp_doc_keyed_name" type="Document">
<Item type="Document" …>
... all document STANDARD properties ...
</Item>
</xp-document>
</Item>
Query:
<Item type="Part" select="xp-document(@defined_as, $value(*, xp-*))" />
Return:
<Item type="Part" id="%Part.ID%" …>
<xp-document defined_as="class" keyed_name="xp_doc_keyed_name" type="Document">
<Item type="Document" …>
... all document STANDARD properties ...
... all document xProperties ...
</Item>
</xp-document>
</Item>
```