ASO APIs - Working With Normalized Attributes

Purpose

This document will detail how to use ASO endpoints to create, retrieve, update, use and delete normalized attributes.

What Are Normalized Attributes

In ASO there are two different ways you can normalize attributes via the API. You can normalize attributes of value-type `string` and you can create attributes of value-type `select-one` which are also normalized. When select-one attributes are first created they must be given a set of values – these select-one values are used to validate any subsequent values entered for this attribute. A normalized string attribute does not require a predefined list of values – rather, every time an attribute value is entered for a normalized string attribute that value is added to the attribute's list of normalized values. This list of normalized values can then be used within ASO to create scenario rule groups, rule scopes, etc. Normalized values are not used for attribute value input validation however like select-one values.

Select-One Attributes

Now we will look at the creation, retrieval, usage and updating of select-one attributes.

Creation

In order to define a select-one attribute the available select-one values for the attribute must also be specified at the time the attribute is created. To create an attribute the ASO **POST Attributes** endpoint will be used. You can find the full documentation for that endpoint here. Before we look at the POST request, let's look at the JSON that will be sent in the request entity-body that will define the new select-one attribute. Note the population of the `selectOneValues` field in the request entity-body in the example JSON.

POST Attributes entity request-body

```
[
  {
    "attributeCode": "a1",
    "name": "Sample Supplier Attribute",
    "description": "Sample supplier select-one attr",
    "attributeType": "supplier",
    "type": "select-one",
    "selectOneValues": [
      "something",
      "something else",
      "again",
      "and again"
    ],
    "isRequired": false,
    "visibleToBidder": true,
    "visibleInManagement": true,
    "namePlural": "Sample Supplier Attributes",
    "nameMix": "Sample Supplier Attribute(s)",
    "isHistoric": false,
    "isLocked": false
  }
]
```

Now let's look at the curl request that will use the above JSON to actually create a new select-one attribute.

POST Attributes request

```
curl -X POST \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Content-Type: application/vnd.sciquest.com.ees+json' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 -d '[{"attributeCode":"a1","name":"Sample Supplier Attribute","description":"Sample supplier select-one attr","
attributeType":"supplier","type":"select-one","selectOneValues":["something","something else","again","and
again"],"isRequired":false,"visibleToBidder":true,"visibleInManagement":true,"namePlural":"Sample Supplier
Attributes","nameMix":"Sample Supplier Attribute(s)","isHistoric":false,"isLocked":false}]' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/apiAttributes"
```

The success response from the request:

POST Attributes response – success

```
{
  "statuses": {
    "SUCCESS": [
      {
        "attributeId": "75",
        "attributeCode": "a1"
      }
    ]
  }
}
```

Retrieval

Once one or more attributes have been successfully created you can retrieve them with various get attribute endpoints. Below we will use the **GET Attribute** endpoint which takes an attribute ID and returns a single attribute. You can find full documentation for that endpoint here. Let's look at a curl request to retrieve our newly created select-one attribute.

GET Attribute request

```
curl -X GET \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/apiAttribute/75"
```

Success response:

GET Attribute response


```
{
  "attributeId": 75,
  "name": "Sample Supplier Attribute",
  "description": "Sample supplier select-one attr",
  "attributeType": "supplier",
  "type": "select-one",
  "apiField": null,
  "isNormalized": true,
  "selectOneValues": [
    null,
    "again",
    "and again",
    "something else",
    "something"
  ],
  "fractionalPrecision": null,
  "isRequired": false,
  "lowValue": null,
  "highValue": null,
  "visibleToBidder": true,
  "visibleInManagement": true,
  "namePlural": null,
  "nameMix": null,
  "isHistoric": false,
  "isLocked": false
}
```


You can see the newly created select-one attribute (attribute 75) was returned in the response. Note that the `selectOneValues` array not only contains the pre-populated values from the POST request, but also the value `null` – all `selectOneValues` will contain a default null value in addition to the pre-defined values. Also note the `isNormalized` field is set to `true` in the response data – this is not necessary to set when creating a select-one attribute and will default to true implicitly, it is however used explicitly when creating normalized string attributes as we will see later.

Usage

Now that we have created a new select-one attribute let's look at how to actually use it. It was created it as a supplier attribute, so we will call the **POST Suppliers** endpoint to send new supplier entity data including a value for our new select-one attribute. You can find full documentation for the **POST Suppliers** endpoint here.

First we will construct the JSON request entity-body that we will be sending in the POST request – this comprises the definition of our new supplier.

POST Suppliers request entity-body

```
[
  {
    "supplierCode": "supp-1",
    "name": "Sample Supplier 1",
    "street1": "218 Murmur St.",
    "city": "Pittsburgh",
    "state": "PA",
    "postalCode": "15212",
    "countryCode": "US",
    "valid": true,
    "attributes": [
      {
        "id": "75",
        "value": "something else"
      }
    ]
  }
]
```

Now let's look at a curl request for **POST Suppliers** using the above request entity-body.

POST Suppliers request #1

```
curl -X POST \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Content-Type: application/vnd.sciquest.com.ees+json' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 -d '[{"supplierCode":"supp-1","name":"Sample Supplier 1","street1":"218 Murmur St.","city":"Pittsburgh","
state":"PA","postalCode":"15212","countryCode":"US","valid":true,"attributes":[{"id":"75","value":"something
else"}]}]' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/user/155375/apiSuppliers"
```

The success response from the request:

POST Suppliers response – success

```
{
  "statuses": {
    "SUCCESS": [
      {
        "supplierId": "20",
        "supplierCode": "supp-1"
      }
    ]
  }
}
```

If you attempt to submit a value that is not represented in the `selectOneValues` array, you will receive the following JSON detailing the error.

POST Suppliers request #2 (bad select-one value)

```
curl -X POST \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Content-Type: application/vnd.sciquest.com.ees+json' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 -d '[{"supplierCode":"supp-2","name":"Sample Supplier 2","street1":"218 Murmur St.","city":"Pittsburgh","
state":"PA","postalCode":"15212","countryCode":"US","valid":true,"attributes":[{"id":"75","value":"something
different"}]}]' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/user/155375/apiSuppliers"
```

Bad select-one value response:

POST Suppliers response – select-one value not found

```
{
  "statuses": {
    "ATTR_ERR_SELECT_ONE_NOT_FOUND": [
      {
        "supplierCode": "supp-2",
        "id": "75",
        "value": "something different"
      }
    ]
  }
}
```

Updating

It is possible to update the `selectOneValues` array for an existing select-one attribute. This is done through the **PATCH Attribute** and **PATCH Attributes** endpoints – these PATCH endpoints will let you change several properties of existing attributes and full documentation for those endpoints can be found here (**PATCH Attribute**) and here (**PATCH Attributes**). It is important to note when updating `selectOneValues` through the PATCH endpoints that the values will be completely replaced by the newly specified values. Let's look at an example using attribute 75 that we created earlier. First we will create the JSON request entity-body that will be sent in the **PATCH Attribute** request. With a PATCH request you only need to specify the fields that you want to change – in this case we are only going to update the `selectOneValues` field.

PATCH Attribute request entity-body

```
{
  "selectOneValues": [
    "a",
    "new",
    "set",
    "of",
    "values"
  ]
}
```

Now let's look at the curl request that will use the above JSON to actually update `selectOneValues`.

PATCH Attribute request

```
curl -X PATCH \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Content-Type: application/vnd.sciquest.com.ees+json' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 -d '{"selectOneValues":["a","new","set","of","values"]}' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/apiAttribute/75"
```

The success response from the request:

PATCH Attribute response – success


```
{
  "statuses": {
    "SUCCESS": [
      {
        "attributeId": "75"
      }
    ]
  }
}
```


Now if we retrieve the attribute again with **GET Attribute**, we can see that the `selectOneValues` field has been updated.

GET Attribute request

```
curl -X GET \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/apiAttribute/75"
```

The success response from the request, with the `selectOneValues` field updated:

GET Attribute response


```
{
  "attributeId": 75,
  "name": "Sample Supplier Attribute",
  "description": "Sample supplier select-one attr",
  "attributeType": "supplier",
  "type": "select-one",
  "apiField": null,
  "isNormalized": true,
  "selectOneValues": [
    null,
    "a",
    "new",
    "set",
    "of",
    "values"
  ],
  "fractionalPrecision": null,
  "isRequired": false,
  "lowValue": null,
  "highValue": null,
  "visibleToBidder": true,
  "visibleInManagement": true,
  "namePlural": null,
  "nameMix": null,
  "isHistoric": false,
  "isLocked": false
}
```


Normalized String Attributes

Now we will look at the creation, retrieval, usage and updating of normalized string attributes.

Creation

In order to define a normalized string attribute the required `type` field must be set to `string` and the optional `isNormalized` field for the attribute must be specified and set to `true` at the time the attribute is created. To create the attribute the ASO **POST Attributes** endpoint will be used. You can find the full documentation for that endpoint here. The following is a curl example invoking the **POST Attributes** endpoint to create a normalized string attribute. Before we actually show the curl request, let's look at the JSON that will be sent in the request entity-body that will define the new normalized string attribute. Note the the presence of the `isNormalized` field in the request entity-body in the example JSON.

POST Attributes entity request-body

```
[
  {
    "attributeCode": "a2",
    "name": "Normalized Supplier Attribute",
    "description": "Sample supplier normalized string attr",
    "attributeType": "supplier",
    "type": "string",
    "isNormalized": true,
    "isRequired": false,
    "visibleToBidder": true,
    "visibleInManagement": true,
    "namePlural": "Normalized Supplier Attributes",
    "nameMix": "Normalized Supplier Attribute(s)",
    "isHistoric": false,
    "isLocked": false
  }
]
```

Now let's look at the curl request that will use the above JSON to actually create a new normalized string attribute.

POST Attributes request

```
curl -X POST \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Content-Type: application/vnd.sciquest.com.ees+json' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 -d '[{"attributeCode":"a2","name":"Normalized Supplier Attribute","description":"Sample supplier normalized
string attr","attributeType":"supplier","type":"string","isNormalized":true,"isRequired":false,"
visibleToBidder":true,"visibleInManagement":true,"namePlural":"Normalized Supplier Attributes","nameMix":"
Normalized Supplier Attribute(s)","isHistoric":false,"isLocked":false}]' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/apiAttributes"
```

The success response from the request:

POST Attributes response – success

```
{
  "statuses": {
    "SUCCESS": [
      {
        "attributeId": "76",
        "attributeCode": "a2"
      }
    ]
  }
}
```

Retrieval

Once one or more attributes have been successfully created you can retrieve them with various get attribute endpoints. Below we will use the **GET Attribute** endpoint which takes an attribute ID and returns a single attribute. You can find full documentation for that endpoint here. Let's look at a curl request to retrieve our newly created normalized string attribute.

GET Attribute request

```
curl -X GET \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/apiAttribute/76"
```

The success response from the request:

GET Attribute response


```
{
  "attributeId": 76,
  "name": "Normalized Supplier Attribute",
  "description": "Sample supplier normalized string attr",
  "attributeType": "supplier",
  "type": "string",
  "apiField": null,
  "isNormalized": true,
  "normalizedValues": [
    null
  ],
  "fractionalPrecision": null,
  "isRequired": false,
  "lowValue": null,
  "highValue": null,
  "visibleToBidder": true,
  "visibleInManagement": true,
  "namePlural": null,
  "nameMix": null,
  "isHistoric": false,
  "isLocked": false
}
```


You can see the newly created normalized string attribute (attribute 76) was returned in the response. Note that the `normalizedValues` array currently only contains the value null – this is because all normalized string attributes still start out with no `normalizedValues` except for the default `null` value which is added automatically. In the *Usage* section below we will see how additional values are added to the `normalizedValued` array.

Usage

Now that we have created a new normalized string attribute let's look at how to actually use it. We created it as a supplier attribute, so we will call the **POST Supplier** endpoint to send new supplier entity data including a value for our new normalized string attribute. You can find full documentation for the **POST Suppliers** endpoint here.

First we will construct the JSON request entity-body that we will be sending in the POST request – this comprises the definition of our new supplier.

POST Suppliers request entity-body

```
[
  {
    "supplierCode": "supp-2",
    "name": "Sample Supplier 2",
    "street1": "222 Hangar St.",
    "city": "Pittsburgh",
    "state": "PA",
    "postalCode": "15212",
    "countryCode": "US",
    "valid": true,
    "attributes": [
      {
        "id": "76",
        "value": "normalized string value"
      }
    ]
  }
]
```

Now let's look at a curl request for **POST Suppliers** using the above request entity-body.

POST Suppliers request #1

```
curl -X POST \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Content-Type: application/vnd.sciquest.com.ees+json' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 -d '[{"supplierCode":"supp-2","name":"Sample Supplier 2","street1":"222 Hangar St.","city":"Pittsburgh","
state":"PA","postalCode":"15212","countryCode":"US","valid":true,"attributes":[{"id":"76","value":"normalized
string value"}]}]' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/user/155375/apiSuppliers"
```

The success response from the request:

POST Suppliers response – success

```
{
  "statuses": {
    "SUCCESS": [
      {
        "supplierId": "21",
        "supplierCode": "supp-2"
      }
    ]
  }
}
```

Now if we retrieve the attribute again with **GET Attribute**, we can see that the value we added for attribute 76 above ("normalized string value") when POSTing the new supplier, has been added to the `normalizedValues` array for that normalized string attribute.

GET Attribute request

```
curl -X GET \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/apiAttribute/76"
```

The success response from the request, with updated `normalizedValues` field:

GET Attribute response


```
{
  "attributeId": 76,
  "name": "Normalized Supplier Attribute",
  "description": "Sample supplier normalized string attr",
  "attributeType": "supplier",
  "type": "string",
  "apiField": null,
  "isNormalized": true,
  "normalizedValues": [
    null,
    "normalized string value"
  ],
  "fractionalPrecision": null,
  "isRequired": false,
  "lowValue": null,
  "highValue": null,
  "visibleToBidder": true,
  "visibleInManagement": true,
  "namePlural": null,
  "nameMix": null,
  "isHistoric": false,
  "isLocked": false
}
```


Unlike with select-one values which are used for input validation, normalized values can be used in ASO to create scenario rule groups, rule scopes, etc.

Updating

Once a string attribute has been created and specified as normalized or standard by using the isNormalized field during attribute creation it cannot be changed. Other attribute properties can be changed using the PATCH Attribute and PATCH Attributes endpoints, but whether or not an attribute is normalized is determined at creation time by the value of the isNormalized field. Full documentation for PATCH endpoints can be found here (PATCH Attribtue) and here (PATCH Attributes).

Attribute Deletion

All attributes, regardless of what type they are, can be deleted using the **DELETE Attribute** endpoint. You can find full documentation for that endpoint here . Let's look at a curl request to delete one of the attributes we created above.

DELETE Attribute request

```
curl -v -X DELETE \
 -H 'Authorization: Bearer xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' \
 -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
 -H 'Content-Type: application/vnd.sciquest.com.ees+json' \
 -H 'Accept: application/vnd.sciquest.com.ees+json' \
 "https://ees.aso-demo-api.va.jaggaer.com/event/22186/apiAttribute/75"
```

Note that the only response from a successful **DELETE Attribute** request will be an **HTTP 204 No Content** response code.

Summary

That should get you started working with ASO normalized attributes. All current ASO API documentation can be found here. Documentation is updated often. If you have any further questions or concerns do not hesitate to get in touch with your ASO Jaggaer contact.