ASO Apis - Working With Asynchronous Entity Imports (Uploads)

Purpose

Depending on the number of entities uploaded in a given request and the validation and processing required for a given entity type, it is often easy to exceed the best practice request-size or request-time limits for REST endpoints. In order to address this ASO has created an asynchronous entity import API workflow. This document will detail how to use the ASO endpoints that implement this asynchronous workflow.

Workflow

The asynchronous import workflow can be used to upload various entities including rates and bids, and proceeds as follows:

1. **Client Initiates Import Process**
   a. Client initiates asynchronous import process by calling EES endpoint **GET Upload URL**
      i. An **`uploadUrl`** and a **`statusUrl`** are returned in the endpoint response along with url expiration details
         1. The **`uploadUrl`** is a customer specific AWS pre-signed PUT URL used to upload entity data (in JSON format) to a private S3 bucket where it is stored encrypted (AES-256) before it is picked up and processed by ASO
         2. The **`statusUrl`** is an ASO endpoint used to poll the status of the import process
2. **Client Uploads Import File**
   a. Client uploads entities in JSON format to the **`uploadUrl`** received in step 1.i where it is stored encrypted, and queued to await processing
3. ***Asynchronous work (non-client step)***
   a. *The import JSON file is retrieved from the queue, validated, and processed*
   b. *When processing has completed a new AWS pre-signed URL is generated to a private S3 bucket where a JSON result file with the detailed results of the import process is uploaded for retrieval by the client*
4. **Client Polls Status**

    a. Client polls the **`statusUrl`** (**GET Async Status** endpoint) received in step 1.i and waits for the status to change to `completed`
        i. When the **GET Async Status** endpoint returns a status of `completed` the response will also contain a **`resultUrl`** – a pre-signed GET URL (generated in step 3b) pointing to the JSON result file containing all relevant information describing the success or failure for each of the entities sent in the step 2 upload
5. **Client Downloads Result File**
    a. Client downloads the result file using **`resultUrl`** received in step 4.i
6. **Client Processes Result File**
    a. Client processes result file noting successes and failures for each entity imported
        i. If any entities were identified to have failed the import process they should be fixed, and then resent in a new entity import process (starting over with step 1)

Details

In the following section we will take a closer look at each of the endpoints/requests involved in the asynchronous entity import process.

GET Upload URL

To initiate the asynchronous entity import process a client will first call the ASO endpoint **GET Upload URL**. For complete documentation of this endpoint, see GET Upload URL.

Request

GET /event/{event-id}/user/{user-id}/entity/{entity}/uploadUrl

Headers

Authorization	Bearer <ASO-bearer-access-token>
X-API-Key	<ASO-customer-API-key>
Accept	application/vnd.sciquest.com.ees+json

URL Parameters

event-id	ASO entity ID
user-id	ASO user ID
entity	ASO entity – one of [`rate`, `bid`]

Response

Fleld	Type	Description
`uploadUrl`	string	Pre-signed URL used to upload entity import file
`uploadUrlExpiresInSeconds`	number	Seconds remaining before `uploadUrl` expires
`statusUrl`	string	URL for polling status of async entity import process
`statusUrlExpiresInSeconds`	number	Seconds remaining before `statusUrl` expires

200 OK

GET Upload URL


```
{
  "uploadUrl": "https://s3.amazonaws.com/us-east-1.aso.devcurr.api.async-uploads-import
/259_22244_devcurr_rate_0daef4fa-21aa-4575-8462-e57fb286dce8?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20230510T172850Z&X-Amz-SignedHeaders=content-type%3Bhost&X-Amz-Expires=300&X-Amz-
Credential=AKIAXROTDM7T32UWEPUQ%2F20230510%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Signature=7d8a19e16233c21bc03705b65a8888b7722efee0b03f4b81cb42610b7d27c229",
  "uploadUrlExpiresInSeconds": 300,
  "statusUrl": "https://ees.devcurr-api.aso-va.jaggaer.com/asyncStatus
/MTY4MzczOTczMDM3NjoyMjI0NDo1MDowZGFlZjRmYS0yMWFhLTQ1NzUtODQ2Mi1lNTdmYjI4NmRjZTg6dXBsb2Fk",
  "statusUrlExpiresInSeconds": 1800
}
```


Pre-Signed PUT Entity Upload URL (`uploadUrl`)

The **`uploadUrl`** returned in the **GET Upload URL** endpoint response is an AWS pre-signed PUT request that requires a custom Accept header and a JSON entity import file to be sent in the request.

Request

PUT

Headers

Accept	application/vnd.sciquest.com.ees+json

Payload

JSON file containing entities to import.

Payloads are entity-specific and currently only rate entities are supported. Following are the rate entity import model details.

Rate

Field	Description	Type	Parent	Required	Notes
rateCode	Unique code to identify rate	string	root	yes	Max length 75
supplierId	ASO supplier ID	number	root	yes	
itemId	ASO item ID	number	root	yes	
alternate	Alternate indicator	number	root	yes	Initial rate must be 0; subsequent rates cannot exceed max-alternates configured in event
rateCurrency	Rate currency code	string	root	yes	Three character currency code string
contractId	Contract ID	string	root	no	
contractName	Contract name	string	root	no	
rateEffectiveDate	Date rate takes effect	number	root	no	Epoch time in millis (13 digits)
rateExpirationDate	Date rate expires	number	root	no	Epoch time in millis (13 digits)
rateDesignation	Identifies a rate's intended use	one of [`Primary`, `Secondary`]	root	no	Can only choose one of the available values
attributes	List of rate attributes	list	root	no	
id	ASO attribute ID	string	attributes	yes	
value	Attribute value	string	attributes	yes	

Example

The following example will resemble most event rate imports.

Sample Rate Import JSON


```
[
  {
    "rateCode": "rate-630",
    "supplierId": 6,
    "itemId": 3,
    "alternate": 0,
    "rateCurrency": "USD",
    "contractId": "c630",
    "contractName": "Rate-C630",
    "rateEffectiveDate": 1663992000000,
    "rateExpirationDate": 1695528000000,
    "rateDesignation": "Primary",
    "attributes": [
      {
        "id": "17",
        "value": "630.00"
      },
      {
        "id": "42",
        "value": "Pittsburgh"
      },
      {
        "id": "43",
        "value": "Helsinki"
      }
    ]
  },
  ...
  {
    "rateCode": "rate-640",
    "supplierId": 6,
    "itemId": 4,
    "alternate": 0,
    "rateCurrency": "USD",
    "contractId": "c640",
    "contractName": "Rate-C640",
    "rateEffectiveDate": 1663992000000,
    "rateExpirationDate": 1695528000000,
    "rateDesignation": "Primary",
    "attributes": [
      {
        "id": "17",
        "value": "640.00"
      },
      {
        "id": "42",
        "value": "Dublin"
      },
      {
        "id": "43",
        "value": "Stockholm"
      }
    ]
  }
]
```


The only thing that will change from entity to entity and event to event when defining the JSON for an import is the entity's attributes section. In our example above we are submitting values for three different attributes – attributes 17, 42, and 43. In order to determine exactly what attributes are available and required for a specific event and entity, use the ASO **GET Attributes by Type** endpoint.

Response

200 OK

Successful upload of the import file using the pre-signed **`uploadUrl`** results in a **200 OK**.

403 Forbidden

A connection to the pre-signed **`uploadUrl`** *after the URL has expired* – indicated by the `uploadUrlExpiresInSeconds` field – will result in a **403 Forbidden**.

GET Async Status (`statusUrl`)

The **`statusUrl`** returned in the **GET Upload URL** endpoint response is an ASO **GET Async Status** endpoint that is used to poll for the status of the asynchronous import process. For full documentation see **GET Async Status**.

Request

/asyncStatus/{encoded-async-pid}

Headers

Authorization	Bearer <ASO-bearer-access-token>
X-API-Key	<ASO-customer-API-key>
Accept	application/vnd.sciquest.com.ees+json

URL Parameters

encoded-async-pid	Base-64 encoded value that identifies the async process

Response

Field	Type	Description	Notes
`status`	enum('initialized','processing','completed','failed','cancelled')	Async process status	
`resultUrl`	string	Pre-signed URL used to download results file	This field only visible when status is `completed`
`resultUrlExpiresInSeconds`	number	Seconds remaining before **`resultUrl`** expires	This field only visible when status is `completed`

Examples

200 OK

```
{
    "status": "initialized"
}
```

The `initialized` status will be returned until an import file is uploaded *and* begins processing.

200 OK

```
{
    "status": "processing"
}
```

The `processing` status indicates that the asynchronous import is in process and polling (if desired) should continue.

200 OK

```
{
  "status": "completed",
  "resultUrl": "https://s3.amazonaws.com/us-east-1.aso.devcurr.api.async-uploads-result/259_22244_b6e5e4fb-d37b-47db-ae8f-a37bef7e0bf3?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20230511T145312Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAXROTDM7T32UWEPUQ%2F20230511%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=f86a039f050ce703c7843474663d529a47ce2832f6d51f066389d8a389c8ddf6",
  "resultUrlExpiresInSeconds": 596
}
```

The `completed` status indicates that the asynchronous import process has completed and the **`resultUrl`** is a pre-signed GET URL that allows the client to download the waiting result file.

403 Forbidden

Attempt to connect *after the URL has expired* – indicated by the **`statusUrlExpiresInSeconds`** field returned in the initial **GET Upload URL** request – will result in a **403 Forbidden** response.

Pre-Signed GET Result URL (`resultUrl`)

The **`resultUrl`** returned in the **GET Async Status** endpoint response is an AWS pre-signed GET request that gives access to the asynchronous import process result file.

Request

GET

Response

200 OK

Connection to the pre-signed URL will return **200 OK** and a stream of the expected results file.

403 Forbidden

A connection to the pre-signed **`resultUrl`** *after the URL has expired* – indicated by the **`resultUrlExpiresInSeconds`** field – will result in a **403 Forbidden**.

Result File

The result file downloaded from **`resultUrl`** will be in JSON form and follows the established standard of existing ASO POST/PATCH/PUT REST API responses. The basic form of the JSON result is the following:

Result JSON

```
{
  "statuses": {
    "STATUS_CODE": [
      {
        "statusKey": "statusValue"
      }
    ]
  }
}
```

Status Codes

Status Code	Key	Description
***ALL** status codes in this table contain this key*	`{entity}Code`	Entity-specific code used to identity the entity – key labels [`rateCode`, `bidCode`]
`MISSING_REQUIRED_FIELD`	---	The referenced entity is missing a required field
	`field`	Name of the missing/required field
`ALREADY_EXISTS`	---	Entity already exists in this event

	`itemId`	Item ID
	`supplierId`	
	`alternate`	
`ENTITY_NOT_FOUND`	---	Entity referenced in key could not be found
	`supplierId` *or* `itemId`	ID of entity that could not be found
`FOR_FIRST_RATE_0_ALTERNATE_REQUIRED`	---	The initial rate for a given supplier/item combination must be alternate 0.
	`field`	Value will always be set to '`alternate`'
`EXCEEDED_MAX_ALTERNATES`	---	
	`maxAlternates`	
`ALTERNATE_OUT_OF_RANGE`	---	
	`field`	Value will always be set to '`alternate`'
`EXCEEDED_CHARACTER_LIMIT`	---	The referenced entity contains a value that exceeds a field's established character limit
	`field`	Name of the field that exceeded the character limit
	`value`	Number of characters attempted
	`limit`	Character limit of the field
***ALL** `ATTR_ERR_*` status codes in this table **also** contain this key*	`id`	Attribute ID
`ATTR_ERR_INVALID`	---	The referenced entity contains an invalid attribute definition
`ATTR_ERR_UNKNOWN`	---	The referenced entity contains an unknown attribute ID
`ATTR_ERR_REQUIRED_MISSING`	---	The referenced entity is missing a required attribute
`ATTR_ERR_REQUIRED_NULL`	---	The referenced entity is passing NULL for a required attribute
`ATTR_ERR_NUMERICAL_HIGH_RANGE`	---	The referenced entity contains an attribute value that exceeds the attributes configured highValue
	`value`	Supplied value
	`highValue`	Attribute configured highValue ceiling
`ATTR_ERR_NUMERICAL_LOW_RANGE`	---	The referenced entity contains an attribute value that falls below the attributes configured lowValue.
	`value`	Supplied value
	`lowValue`	Attribute configured lowValue floor
`ATTR_ERR_NUMERICAL_PRECISION`	---	The referenced entity contains an attribute value that does not match the attribute's configured fractionalPrecision
	`value`	Supplied value
	`actualPrecision`	Precision of the value supplied in entity-body
	`expectedPrecision`	Attribute configured fractionalPrecision
`ATTR_ERR_NUMERICAL_FORMAT`	---	The referenced entity contains an attribute value that does not fit the appropriate numerical format
	`value`	Supplied value
`ATTR_ERR_DATE_TIME_FORMAT`	---	The referenced entity contains an attribute value that does not fit the appropriate date format
	`value`	Supplied value
	`format`	Expected format

ATTR_ERR_BOOLEAN_NOT_FOUND	---	The referenced entity contains an attribute value that does not fit the appropriate boolean format
	value	Supplied value
	expected	Expected values
ATTR_ERR_CURRENCY_NOT_FOUND	---	The referenced entity contains an attribute value that does not fit the appropriate currency code format
	value	Supplied value
ATTR_ERR_SELECT_ONE_NOT_FOUND	---	The referenced entity contains an attribute value that does match a valid normalized string value
	value	Supplied value

Examples

Following are result file examples.

Success: Single Entity

```
{
  "statuses": {
    "SUCCESS": [
      {
        "rateId": "19",
        "rateCode": "rate-330"
      }
    ]
  }
}
```

As you can see in the case of a successful import, the newly created ID of the entity will be returned along with the 'code' that was supplied at the time of import. In most cases the client will be importing multiple entities in a single import file, and so the result file must reflect a result for each entity imported.

Success: Multi Entity

```
{
  "statuses": {
    "SUCCESS": [
      {
        "rateId": "20",
        "rateCode": "rate-331"
      },
      {
        "rateId": "21",
        "rateCode": "rate-340"
      },
      {
        "rateId": "22",
        "rateCode": "rate-341"
      }
    ]
  }
}
```

Of course not every entity will be successfully processed. Often, and especially when many are being imported in a single file, entities may fail validation. If this occurs you will see the reasons for these failures in the result file.

Fail: Single Entity


```
{
  "statuses": {
    "ENTITY_NOT_FOUND": [
      {
        "itemId": "3",
        "rateCode": "rate-630"
      }
    ]
  }
}
```


Here we can see a rate was attempted to be imported for an item with ID 3 but that item has not been found in the associated event, so an appropriate status code is returned noting the item ID as well as the rate 'code' so that the client can attempt to amend the rate and re-import.

Failure: Multi Entity


```
{
  "statuses": {
    "ENTITY_NOT_FOUND": [
      {
        "itemId": "3",
        "rateCode": "rate-630"
      }
    ],
    "ALREADY_EXISTS": [
      {
        "itemId": "4",
        "supplierId": "6",
        "alternate": "0",
        "rateCode": "rate-460"
      }
    ],
    "ATTR_ERR_REQUIRED_MISSING": [
      {
        "id": "42",
        "rateCode": "rate-140"
      },
      {
        "id": "43",
        "rateCode": "rate-140"
      }
    ]
  }
}
```


In the above result file, there are several issues being reported:

1. rate-630 was imported for item 3, but that item does not exist (`ENTITY_NOT_FOUND`)
2. rate-460 was imported by supplier 6 on item 4 as alternate 0, but that combination has already had a rate accepted for it (`ALREADY_EXISTS`)
3. rate-140 was missing values for two required attributed – attribute 42, and attribute 42 (`ATTR_ERR_REQUIRED_MISSING`)

Mixed: Multi Entity


```
{
  "statuses": {
    "SUCCESS": [
      {
        "rateId": "7",
        "rateCode": "rate-710"
      },
      {
        "rateId": "8",
        "rateCode": "rate-720"
      },
      {
        "rateId": "9",
        "rateCode": "rate-730"
      },
      {
        "rateId": "10",
        "rateCode": "rate-731"
      }
    ],
    "ATTR_ERR_LOCATION_NOT_FOUND": [
      {
        "id": "42",
        "value": "Mars",
        "rateCode": "rate-661"
      },
      {
        "id": "43",
        "value": "Jupiter",
        "rateCode": "rate-661"
      }
    ],
    "ALREADY_EXISTS": [
      {
        "itemId": "5",
        "supplierId": "6",
        "alternate": "0",
        "rateCode": "rate-560"
      }
    ]
  }
}
```


Finally, the most likely result you will see is a mixed result file with many successfully processed rates along with a few failures that may need to be corrected and re-imported.

In Practice

The following section will walk-through an example Rate upload with the asynchronous entity import process using simple curl statements.

Initiate the Asynchronous Rate Import Process

Out initial step is to call the **GET Upload URL** endpoint, this initiates the asynchronous import process and returns two URLs that we use in our next steps:

Call: GET Upload URL

```
$ curl -X GET \
>  -H 'Authorization: Bearer 9db78c6c-882f-403d-9857-f5518c1b3dfe' \
>  -H 'Accept: application/vnd.sciquest.com.ees+json' \
>  "http://ees.devcurr-api.aso-va.jaggaer.com/event/22195/user/155375/entity/rate/uploadUrl"
```

Response:

200 OK

GET Upload URL Reponse


```
{
  "uploadUrl": "https://s3.amazonaws.com/us-east-1.aso.devcurr.api.async-uploads-import
/259_22195_devcurr_rate_f4ffdf1b-0a76-49bf-ac00-0307373d7f4b?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20230512T140921Z&X-Amz-SignedHeaders=content-type%3Bhost&X-Amz-Expires=300&X-Amz-
Credential=AKIAXROTDM7T32UWEPUQ%2F20230512%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Signature=fd31b01589966bb1a2b41dca1aa9886dcc5abceb76d917cfb2acacdbd734313e",
  "uploadUrlExpiresInSeconds": 300,
  "statusUrl": "https://ees.devcurr-api.aso-va.jaggaer.com/asyncStatus
/MTY4MzkwMTE5MDQ2NDoyMjE5NTo2OToxNDZiNjE4ZC03MTBkLTQ0MzUtOWIwYi04M2I2Njg0NTJkMDI6dXBsb2Fk",
  "statusUrlExpiresInSeconds": 1800
}
```


Upload the Rate Import File

Using the **`uploadUrl`** received in the **GET Upload URL** response we can now upload our rate import file:

Call: Pre-Signed PUT File Upload

```
$ curl -X PUT \
>  -H 'Content-Type: application/vnd.sciquest.com.ees+json' \
>  -d "@rates-22195-1" \
>  'https://s3.amazonaws.com/us-east-1.aso.devcurr.api.async-uploads-import/259_22195_devcurr_rate_642c1094-
8f51-42de-a32e-b2fef784381f?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20230512T141550Z&X-Amz-
SignedHeaders=content-type%3Bhost&X-Amz-Expires=300&X-Amz-Credential=AKIAXROTDM7T32UWEPUQ%2F20230512%2Fus-east-
1%2Fs3%2Faws4_request&X-Amz-Signature=99b0b96d33f2f4e97ab2e6cc4fd5ae6415b308e049f2c995fbb5a69855f4474b'
```

Contents of rate import file `rates-22195-1`:

rates-22195-1


```
[
  {
    "rateCode": "rate-340",
    "supplierId": 3,
    "itemId": 4,
    "alternate": 0,
    "rateCurrency": "USD",
    "contractId": "R340",
    "contractName": "Rate-340",
    "rateEffectiveDate": 1663992000000,
    "rateExpirationDate": 1695528000000,
    "rateDesignation": "Primary",
    "attributes": [
      {
        "id": "17",
        "value": "340.00"
      },
      {
        "id": "41",
        "value": "Pittsburgh"
      }
    ]
  }
]
```


Response:

200 OK

Check the Status of the Import Process

Now that the import file has been uploaded, using the **`statusUrl`** received in the **GET Upload URL** response we can check the status of our import process:

Call: GET Async Status

```
$ curl -X GET \
>  -H 'Authorization: Bearer 9db78c6c-882f-403d-9857-f5518c1b3dfe' \
>  -H 'Accept: application/vnd.sciquest.com.ees+json' \
>  "http://ees.devcurr-api.aso-va.jaggaer.com/asyncStatus
/MTY4MzkwMTE5MDQ2NDoyMjE5NTo2OToxNDZiNjE4ZC03MTBkLTQ0MzUtOWIwYi04M2I2Njg0NTJkMDI6dXBsb2Fk"
```

Response:

200 OK

GET Async Status Reponse

```
{
  "status": "initialized"
}
```

We have received an `initialized` status indicating that the process has not started yet. If we continue polling with the **`statusUrl`** we should see the status update:

GET Async Status Reponse

```
{
  "status": "processing"
}
```

After a few more calls to **GET Async Status** (**`statusUrl`**) we see that the status has changed to `processing`. Finally, we will continue polling and look for a `completed` status:

GET Async Status Reponse

```
{
  "status": "completed",
  "resultUrl": "https://s3.amazonaws.com/us-east-1.aso.devcurr.api.async-uploads-result/259_22195_146b618d-710d-
4435-9b0b-83b668452d02?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20230512T142118Z&X-Amz-SignedHeaders=host&X-
Amz-Expires=600&X-Amz-Credential=AKIAXROTDM7T32UWEPUQ%2F20230512%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Signature=dc0126baf811969820866c4bd80e12620cbc7d5eb1c472732aad4657dab3506e",
  "resultUrlExpiresInSeconds": 583
}
```

With the `completed` status we also receive a **`resultUrl`** which is how we will receive the rate import process result file.

Download the Result File

With one last call to the **`resultUrl`** we are able to retrieve the result file for the rate import process:

Call: Pre-signed GET Result File

```
curl -X GET \
 'https://s3.amazonaws.com/us-east-1.aso.devcurr.api.async-uploads-result/259_22195_146b618d-710d-4435-9b0b-
83b668452d02?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20230512T142118Z&X-Amz-SignedHeaders=host&X-Amz-
Expires=600&X-Amz-Credential=AKIAXROTDM7T32UWEPUQ%2F20230512%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Signature=dc0126baf811969820866c4bd80e12620cbc7d5eb1c472732aad4657dab3506e'
```

Response:

200 OK

Result File (contents)


```
{
  "statuses": {
    "SUCCESS": [
      {
        "rateId": "16",
        "rateCode": "rate-340"
      }
    ]
  }
}
```


Summary

That should give you a good start towards working with ASO's asynchronous entity imports. All current ASO API documentation can be found at http://docs.aso.engineering/. Documentation is updated often. If you have any further questions or concerns do not hesitate to get in touch with your ASO Jaggaer contact.