



The URL that is returned to the calling client from an ASO asynchronous endpoint invokes the **GET Async Status** endpoint and allows the client to poll the status of the corresponding asynchronous process.

## GET Async Status

Endpoint that returns the status of an asynchronous process. For full documentation follow [GET Async Status](#).

### Request Details

#### Headers

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

### Response Details

Field	Type	Description	Notes
status	enum('processing','completed','failed','cancelled')	Async process status	
resultUrl	string	Pre-signed URL used to download resulting 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": "processing"
}
```

Obviously the 'processing' status indicates that the asynchronous process is still processing and polling (if desired) should continue.

#### 200 OK

```
{
  "status": "completed",
  "resultUrl": "https://s3.us-east-2.amazonaws.com/us-east-2.aso.ga.api.async-downloads/XXX_XXXXX_50a0a503-a029-4316-894f-3a7fdcf07ab2?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210716T154359Z&X-Amz-SignedHeaders=host&X-Amz-Expires=300&X-Amz-Credential=AKIA5HYMQXHER7KSSZTN%2F20210716%2Fus-east-2%2Fs3%2Faws4_request&X-Amz-Signature=7cdb913ef900e2eef7b5f6bed6a592d056e52d611053976a11e3175d92eb674",
  "resultUrlExpiresInSeconds": 289
}
```

The 'completed' status indicates that the asynchronous process has completed and the downloadUrl is a pre-signed URL that allows the client to download the waiting file.

#### 403 Forbidden

Attempt to connect after the URL has expired – indicated by the expiresInSeconds field returned in the initial /async request – will result in a **403 Forbidden** response.

### Pre-Signed GET URL (resultUrl)

The resultUrl field returned by a 'completed' **GET Async Status** response contains an AWS S3 pre-signed URL.

#### 200 OK

Connection to the pre-signed URL will return **200 OK** and a stream of the expected file. A detailed description of each endpoint's entity response data can be found in each endpoint's documentation at <http://docs.aso.engineering/>.

### 403 Forbidden

A connection to the pre-signed URL *after the URL has expired* – indicated by the `resultUrlExpiresInSeconds` field returned in a 'completed' **GET Async Status** response – will result in a **403 Forbidden**.

## In Practice

### Programmatic Example

The following is a simple Java example of how to poll and download the file once it is available.

#### Model

##### Model

```
public class AsyncStatus {
    private String status;
    private String resultUrl;
    private Integer resultUrlExpiresInSeconds;

    public AsyncStatus() { }

    public String getStatus() { return status; }
    public void setStatus(String status) { this.status = status; }

    public String getResultUrl() { return resultUrl; }
    public void setResultUrl(String resultUrl) { this.resultUrl = resultUrl; }

    public Integer getResultUrlExpiresInSeconds() { return resultUrlExpiresInSeconds; }
    public void setResultUrlExpiresInSeconds(Integer resultUrlExpiresInSeconds) { this.
resultUrlExpiresInSeconds = resultUrlExpiresInSeconds; }
}
```

This POJO can be used to serialize the JSON from the **GET Async Status** response.

#### Imports

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

import com.fasterxml.jackson.databind.ObjectMapper;
```

We are using the Jackson libraries to serialize/deserialize our JSON.

#### Code

Use the **GET Async Status** URL returned by any ASO asynchronous endpoint to poll the asynchronous process status until the process has completed.

## Polling

```
System.out.println("=== Polling Async Status ===");
URL url = new URL(asyncStatusUrlStr);
do {
    connection = (URLConnection) url.openConnection();
    connection.setRequestProperty("Accept", "application/vnd.scicquest.com.ees+json");
    connection.setRequestProperty("Authorization", "Bearer "+bearerAccessToken);
    connection.setRequestProperty("X-API-Key", apiKey);
    int responseCode = connection.getResponseCode();
    if (responseCode == 200) {
        StringBuilder responseBuilder = new StringBuilder();
        try (InputStream content = connection.getInputStream()) {
            BufferedReader in = new BufferedReader(new InputStreamReader(content));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                responseBuilder.append(inputLine);
            }
        }
        // use jackson to serialize response
        asyncStatus = mapper.readValue(responseBuilder.toString(), AsyncStatus.class);
        System.out.println("Async status: ["+response.getStatus()+"]");
        if (asyncStatus.getStatus().equals("processing")) {
            // sleep for desired interval because status is still 'processing'
            System.out.println("Sleeping "+intervalSeconds+"s");
            Thread.sleep(intervalSeconds * 1000);
        }
    }
    elapsedMillis = System.currentTimeMillis() - startMillis;
    // continue polling while we are getting 200's AND still processing AND under the configured maximum time
} while (connection.getResponseCode() == 200 && asyncStatus.getStatus().equals("processing") && elapsedMillis <
(maxSeconds * 1000));
```

The above logic will poll the **GET Async Status** endpoint until it finishes processing, fails, or the total polling time eclipses the configured maximum time.

Once the **GET Async Status** endpoint returns a 'completed' status the resulting response will also contain a `resultUrl` – a presigned-url URL that can be used to download the expected file.

```

switch (asyncStatus.getStatus()) {
    case "completed":
        // handle completed
        System.out.println("Downloading completed async file...");
        url = new URL(asyncStatus.getResultUrl());
        connection = (URLConnection) url.openConnection();
        int responseCode = connection.getResponseCode();
        if (responseCode == 200) {
            System.out.println("Returned response (200): ");
            // download file
            try (InputStream content = connection.getInputStream()) {
                BufferedReader in = new BufferedReader(new InputStreamReader(content));
                String inputLine;
                while ((inputLine = in.readLine()) != null) {
                    // write file to std out
                    System.out.println(inputLine);
                }
            }
        } else if (responseCode == 403) {
            System.out.println("File no longer available (403)");
        } else if (responseCode == 404) {
            System.out.println("WARNING: File not yet available (404); Download
ABORTED");
        } else {
            System.out.println("Unexpected response code: "+responseCode);
        }
        break;
    case "cancelled":
        // handle cancelled...
        break;
    case "failed":
        // handle failed...
        break;
}

```

In this example, for a 'completed' AsyncStatus object we connect to the pre-signed URL (resultUrl field) and write the downloaded file to standard out.

## Summary

That should get you started working with ASO asynchronous endpoints. 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.