JULY 16, 2026

GSCCCA IMAGE API USER GUIDE

GEORGIA SUPERIOR COURT CLERKS' COOPERATIVE AUTHORITY

HTTPS://WWW.GSCCCA.ORG

REVISION HISTORY

Revision	Description	Date
1	Initial Version	03/25/2026

CONTENTS

INFORMATION FOR IMPLEMENTERS

The Georgia Superior Court Clerks' Cooperative Authority (GSCCCA) offers a public REST (Representational State Transfer) API that allows for the submission, retrieval, and reporting of images. Vendors will send images using the API. Images will be validated, and a report can be created.

ASSUMPTIONS

This is a technical document intended for developers and service providers who wish to upload deed, plat, and lien images using the Image API. Anyone reading this document should be well-versed in HTTP Request methods, reading and serializing JSON, and connecting to a REST web service to make calls and process responses. Expectations are that API callers will be familiar with the Image API Image Specification.

AUTHENTICATION

The GSCCCA Image API requires authentication through the GSCCCA Identity Server. When your Image API account is activated, you will be assigned a client identifier and secret that will be used for authentication. For information on how to authenticate against various GSCCCA APIs, you will need to read the documentation on https://www.gsccca.org/learn/certification-programs/real-property-system-resources. It is titled GSCCCA Identity Server Authentication Guide.

SUPPORT/CONTACT

For all GSCCCA Image API related questions, please contact our customer support at (800) 304-5174 or email help@gsccca.org.

WHERE DO I CONNECT?

The Image API is a REST service that provides various web methods to allow users to programmatically submit and report on their images via web service calls.

TEST SERVICE ENDPOINTS

For testing purposes, the GSCCCA provides a development test environment. Access to the GSCCCA test website is restricted. Please email help@gsccca.org to request access.

The public test endpoints are all located at this address:

- https://apisstg.gsccca.org/imaging/v1/

Test OpenAPI Specification (OAS): https://apisstg.gsccca.org/imaging/v1/api-docs/index.html

PRODUCTION SERVICE ENDPOINTS

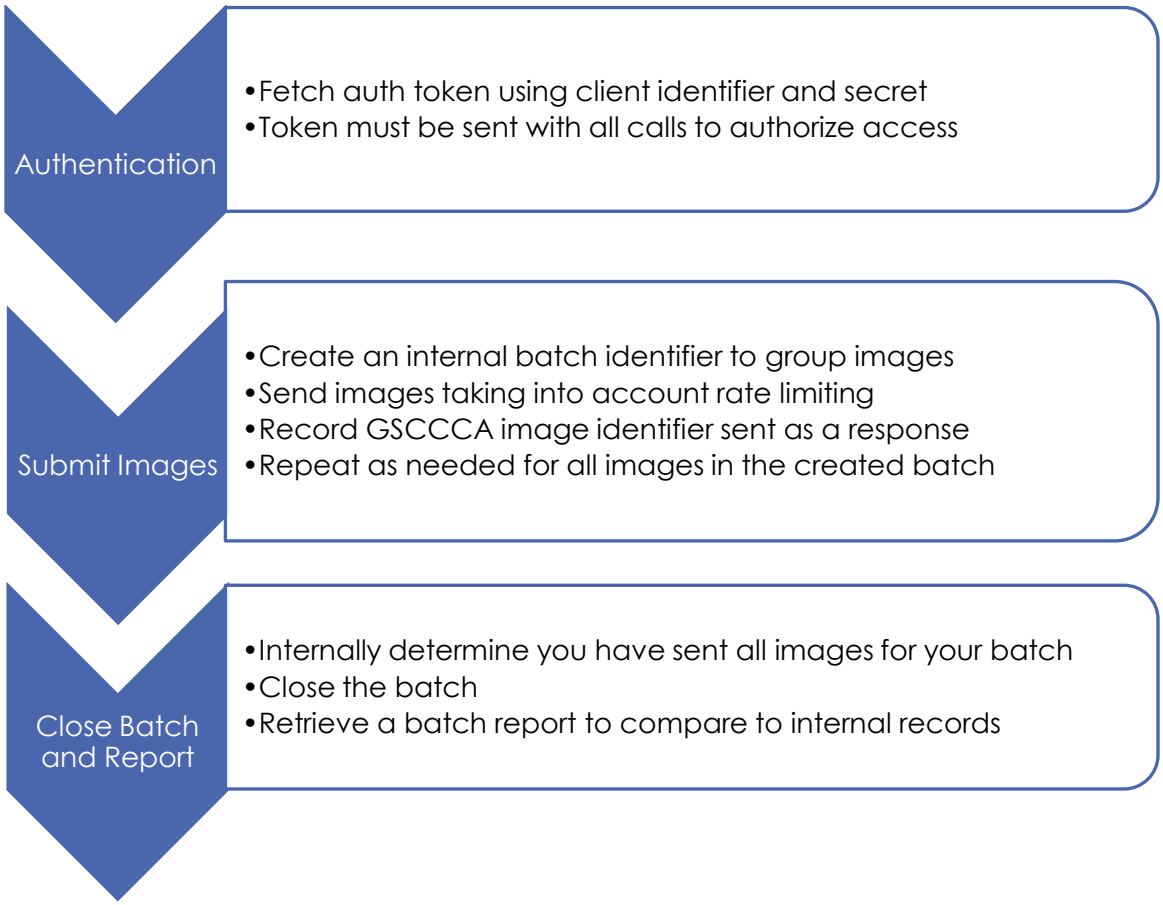
Once you have sufficiently tested and are ready to begin submitting images, the production endpoints are all located at this address.

The public Image API endpoint is here:

- https://apis.gsccca.org/imaging/v1/

IMAGE API WORKFLOW

Below is the basic workflow of the overall image submission process. This workflow will allow users to submit a series of images in a batch (see below) and report on that batch. Once the workflow is complete, those images are all submitted and in the GSCCCA Imaging system.



WHAT IS A BATCH?

A batch is a logical grouping mechanism that helps users organize images being sent.

A given batch is assigned a name by the user or system interfacing with the GSCCCA Image API. This name can be any alpha-numeric of 25 characters or less. The batch is transmitted to the GSCCCA either explicitly or implicitly when submitting an image with a new batch ID.

It is subsequently closed by the user when all images in the group have been sent by call to close the batch. Calls can also be made to reporting endpoints to receive summary and detailed JSON and optionally to have HTML reports emailed to specified email addresses. Best practice is to create and close batches when starting and sending image groups to prevent abandoned batches.

A batch must be closed to query its report. A closed batch cannot include further submitted images. This ensures the immutability of any reported batch. Multiple batches can be open and used in submission at the same time, but developers must be aware of rate limiting settings on the system. Batches are limited and must be closed when they reach 30,000 images.

DATA AND IMAGE REQUIREMENTS

Images should follow the guidelines set out in the Real Property Recording Systems - Image API Image Specification. The details of the specification can be found on the GSCCCA website.

https://www.gsccca.org/learn/certification-programs/real-property-system-resources

It is expected for developers to be familiar with the specifications in the document. For a list of validation error results that may be generated when submitting an image, see Appendix A.

OPENAPI SPECIFICATION (OAS)

Full details about each call, including response schema and examples, are provided on the OAS page. This page is updated regularly and contains the most detailed and current information.

https://apis.gsccca.org/imaging/v1/api-docs/index.html

WHAT ENDPOINTS ARE PROVIDED?

All web methods made available for API callers to allow the submission and reporting of images are included in the OAS referenced in the previous section. Each endpoint definition includes a name, a description of what the web method is used for, and all required and optional parameters. A brief description of the response is included along with any remarks that explain how to use the given web method.

RESPONSE CODES AND CONTENT

Calls to all endpoints will return with an HTTP Status Code indicating a general response type. They will also include JSON as detailed by the endpoint itself. Below are the response codes used throughout the GSCCCA Image API. More details are included with each endpoint.

HTTP CODE EXAMPLES

200	OK	No errors encountered. Success.
401	Unauthorized	Included auth token does not allow access to this endpoint.
400	Bad Request	Client input was not valid. JSON will contain more information.
403	Forbidden	Client has permission to access the API but does not have permission to perform the requested action.
404	Not Found	A resource that was requested does not exist.
422	Unprocessable Entity	Image Validation Error
429	Too Many Requests	Client has sent too many calls in a given timeframe
500	Internal Server Error	The GSCCCA system encountered an error.

JSON ERROR RESPONSE EXAMPLE


```
{
 "type": "https://tools.ietf.org/html/rfc7231#section-6.5.8",
 "title": "Validation Error",
 "status": 422,
 "detail": "Image with image identifier: 6044 was Rejected and didn't pass validation
 -  INVALID_COLOR_DEPTH. Current Settings - DPI: 200, Color Depth: 256 Colors, Compression: CCITT Fax 4.",
 "traceId": "|6b4751a0-478f027b952efab0."
}
```


JSON SUCCESS RESPONSE EXAMPLES

For examples of JSON sent by each of the GSCCCA endpoints, see our OAS section referenced in this documentation.

RATE LIMITING

Calls to all endpoints are rate-limited. Any calls beyond the defined limits return an error response. Callers are expected to limit the number of calls made to the services to avoid rate limitation violations.

Limits given are per second, per minute, and per hour to improve user experience.

APPENDIX A: VALIDATION ERROR CODES

When submitting an image, several validation steps are performed. Any validation failures are returned to the caller. If multiple validation steps fail, they are all returned.

The currently defined validation errors are listed below.

TEXT	DESCRIPTION
NO_SETTINGS_CONFIGURED	No settings were found for account
RENDERABILITY_INVALID	System could not render the image
BOOK_LENGTH_INVALID	Book length is invalid
PAGE_LENGTH_INVALID	Page length is invalid
INVALID_DOC_TYPE	Invalid Document Type
INVALID_COUNTY_CODE	Invalid County Code
FILE_READ_ERROR	Unable to read the File
FILE_SIZE_ZERO	File has no content
ENDINESS_NOT_SET	The Endianness is not set on the image
MAGIC_NUMBER_NOT_SET	Magic number is not set on the image
X_RES_NOT_VALID	X Resolution is not valid
Y_RES_NOT_VALID	Y Resolution is not valid
INVALID_COLOR_DEPTH	Invalid Color Depth
INVALID_COMPRESSION	Invalid Compression
ARTIST_TAG_NOT_SET	Artist tag is not set
DOCNAME_NOT_SET	Document name is not set
DOCNAME_INVALID	Document name is invalid

APPENDIX B: CODE EXAMPLES

Below are various code examples to help developers with core concepts of the GSCCCA Image API. These do not constitute complete code and come with no guarantee that they work on their own. These examples define baseline functionality without proper sanitation, logging, error handling, or user interaction. Be sure to integrate using best practices.

All code examples are in the C# programming language. If you use a different language, look for equivalent calls and utilize the code examples as pseudo-code.

IMAGE CHECKSUM

All calls to `/images POST` must include a checksum in the header. Upon receiving this call, the GSCCCA system will perform its own checksum on the image bytes sent and compare the submitted checksum with the API's generated checksum to ensure a match. This checksum will then be used to check that the submitted image is not a duplicate.

The checksum is a SHA-1 hash of the image byte array in a hexadecimal number format.

```
public static string CalculateChecksum(byte[] img)
{
    // Create SHA1 hasher
    using (var sha1Hasher = SHA1.Create())
    {
        //Compute hash
        var data = sha1Hasher.ComputeHash(img);

        //base 64 encode
        return Convert.ToBase64String(data);
    }
}
```

CALL REST ENDPOINTS

All GSCCCA Image API endpoints are REST based HTTP calls over SSL.

Calling the image API requires a GSCCCA Identity Server access token, which is a JSON Web Token (JWT). The JWT contains authorization information that the API will use to allow or deny access to various calls. For more information on how to request a token, see the "Information for Implementers" section at the top of this document.

Below is a basic example of setting a token on an HttpClient

```
private async Task SetToken(string tokenUrl, HttpClient client)
{
    var request = new HttpRequestMessage();
    request.RequestUri = new Uri(tokenUrl);
    request.Method = HttpMethod.Post;
    request.Content = new FormUrlEncodedContent(new[]
    {
        new KeyValuePair<string, string>("grant_type", "client_credentials"),
        new KeyValuePair<string, string>("scope", "client.image.api"),
        new KeyValuePair<string, string>("client_id", "YOUR_CLIENT_ID_HERE"),
        new KeyValuePair<string, string>("client_secret", "YOUR_CLIENT_SECRET_HERE")
    });
    var response = await client.SendAsync(request);
    var jsonResult = await response.Content.ReadAsStringAsync();
    var result = JsonConvert.DeserializeAnonymousType(jsonResult, new {access_token = ""});
    client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
result.access_token);
}
```

Below is a basic example of calling a GET endpoint. This is a generic, repeatable, format.

```
private async Task<HttpResponseMessage> CallGET(HttpClient client,string url)
{
    //make sure to call SetToken before calling this method
    var result = await client.GetAsync(url);
    //throw an exception if the API returned an error
    result.EnsureSuccessStatusCode();
    return result;
}
```

Here is an example of using the `CallGET()` function to fetch an image with `/images GET`.

```
private async Task GetImage(HttpClient client, string tokenUrl, string imgUrl)
{
    await SetToken(client, tokenUrl);

    // Call /image GET
    var response = await CallGET(client, imgUrl);

    // Parse response
    var bytes = await response.Content.ReadAsByteArrayAsync();

    // Write Bytes
    File.WriteAllBytes(@"C:\tmp\mydownloadedfile.tif", bytes);
}
```

This is an example of submitting a new image

```
//the postUrl will include the batch name
private async Task<string> UploadImage(HttpClient client, string filePath, string postUrl)
{
    //Make sure the token is set on the httpclient before calling this function
    var bytes = File.ReadAllBytes(filePath);
    //get the checksum value using the previously documented function
    var checksum = CalculateChecksum(bytes);
    using (var formData = new MultipartFormDataContent())
    {
        //add the file content to the multipart form request
        formData.Add(new ByteArrayContent(bytes),"File1",Path.GetFileName(filePath));
        var request = new HttpRequestMessage();
        request.RequestUri = new Uri(postUrl);
        request.Method = HttpMethod.Post;
        request.Content = formData;
        //add the checksum header tot he request
        request.Headers.Add("Checksum", checksum);
        //do the POST
        var result = await client.SendAsync(request);
        //throw an exception if the API returned an error
        result.EnsureSuccessStatusCode();
        //return the JSON response
        return await result.Content.ReadAsStringAsync();
    }
}
```