Sopheon Accolade®

Accolade Web API Reference Guide

Version: 17.1

About Sopheon Accolade®

Document Name:	Accolade Web API Reference Guide
Document Version:	1
Software Version:	Sopheon Accolade 17.1
Document Date:	November 2024

Ownership of Software and Documentation

The Sopheon® software described in this documentation is furnished under a license agreement and may be used only in accordance with the terms of that license agreement.

Sopheon Corporation and its associated Sopheon Group companies, including its subsidiaries, its immediate holding company and its ultimate holding company (together, "Sopheon") have created and own all rights to the software and documentation. Licensees of the software have purchased a limited right to use the software in accordance with their license agreement.

Copyright Notice

Trademarks

"Accolade", "Sopheon", and the Sopheon logo are registered trademarks of Sopheon. "Vision Strategist", the Vision Strategist logos, "Idea Lab", and "Process Manager" are trademarks of Sopheon. A more complete list of Sopheon trademarks is available at www.sopheon.com.

"Microsoft", "Windows", "Excel" , "PowerPoint" and "Microsoft Teams" are registered trademarks of Microsoft Corporation. A complete list of Microsoft trademarks is available at www.microsoft.com . "Lotus Notes" is a registered trademark of International Business Machines Corporation. "WinZip" is a registered trademark of WinZip Computing, Inc. "Stage-Gate" is a registered trademark of the Product Development Institute. Other product names mentioned in this Help system may be trademarks of their respective companies and are hereby acknowledged.

"Slack" is a registered trademark of Salesforce Technologies, LLC.

Names of persons or companies and other data contained in examples set forth in this user documentation are fictitious unless otherwise noted.

No Warranty

The technical documentation is being delivered to you AS-IS, and Sopheon makes no warranty as to its accuracy or use. Any use of the technical documentation or the information contained therein is at the risk of the user. Documentation may include technical or other inaccuracies or typographical errors. Sopheon reserves the right to make changes without prior notice. In no circumstances will Sopheon, its agents or employees be liable for any special, consequential or indirect loss or damage arising from any use of or reliance on any materials in this documentation or in the software.

Patents

Aspects of Sopheon software are protected by U.S. Patents 5634051, 6632251, and 6526404; European Patent EP0914637; and by U.K. Patent GB2341252A.

Contents

About this Guide

Welcome to the *Sopheon Accolade Web API Reference Guide*. The Web API allows Accolade users to design and implement customer configuration solutions that integrate with the core Accolade application.

This guide contains instructions for setting up user authentication when creating API calls to access Accolade information, as well as design information for BI data scientists and developers.

This information presented in this guide is intended for use with the Accolade Web API v. 2.5, to be used in conjunction with Sopheon Accolade v.17.1.

Note: Your company may use other components of the Accolade Web API resources for other purposes in your business. This guide discusses only the information you need to access the Accolade Web API resources for use in Accolade configuration. For additional information about the Accolade Data API area, please contact Sopheon Customer Support or refer to the Data API topics in the Accolade application main Help.

Assumptions

This guide assumes you are familiar with Accolade. For more information about Accolade, see the Accolade online Help available from within the main application.

Font Conventions

- This **bold font** is used for important words and the names of the items you need to identify.

  Create a SQL account named “Geneva”, and give this account the **VS_Write** database role.

- `This fixed-width font` is used for examples of code, paths, and URLS.

  `https//:your-server-name:port-number/`

- *This italic font* is used for document names.

- *An italic font* enclosed in brackets shows what information is displayed in this location when the information is changeable, rather than fixed.

  Process Document - Smart Excel <*version*>.xlt

- Blue text indicates a cross-reference link that you can click to take you to that location.

Icon Conventions

💡 - Indicates a tip to assist with Accolade configuration or management.

EXAMPLE - Indicates an example use case to assist with Accolade component configuration.

> **Important!** This is an important statement. Read it carefully before proceeding with an action.

Related Documentation

Sopheon Accolade Web API Reference Guide

Contacting Technical Publications

To send comments and suggestions regarding this document, send email to techpubs@sopheon.com.

Chapter 1

Accolade Web API Overview

An application programming interface (API) is an interface that enables interaction with other software. APIs let programs share information and influence each others' behavior through a "request and response" method of conversation. This conversation is similar to a conversation between people, but with defined rules about the type of communication allowed. APIs can be used both to retrieve and update data.

Accolade as a RESTful API

The Accolade Web API is a private RESTful web service, intended for use by company assets and by our customers. A RESTful API is based on representational state transfer (REST), an architectural style and approach to communications often used in web services development. A RESTful web application exposes information about itself in the form of information about its resources. It also enables the client to take actions on those resources, such as create new resources (i.e. create a new user) or change existing resources (i.e. edit a post).

Accolade’s Web API conforms to the Richardson Maturity Model Level 2. It is implemented with ASP.Net Web API 2.2 with attribute routing, and follows OData v3 conventions. JSON is the media type that is supported for request and response payloads.

Accolade API Versioning

Accolade v.17.1 currently supports Accolade Web API version 2.5. The Accolade Web API is semantic versioned, and the version number is independent from the Accolade version. The version number has the MAJOR.MINOR format and is incremented as follows:

- **MAJOR** version when there are **incompatible** API changes.
- **MINOR** version when functionality is added in a **backwards-compatible** manner.

Accolade API Authentication and Access

Accolade has several areas within its API designed to retrieve the appropriate information based on the application. Before users can make requests, they need to have a method for authenticating the requests. The Accolade API has two methods for verifying requests:

- using authentication tokens to identify and verify the users.
- via a generated API key, which identifies the application making the API call.

Access to the Accolade Web API is available to any developer, and requests are authenticated via the user's Accolade credentials. The Web API areas can be used to create customized configuration solutions for users within your company, such as creating an advanced layout which uses functionality in a non-standard way, or retrieving the results of an Accolade online report for use in process configuration.

Access to the Accolade Data API requires the use of an API key, which can be created by users with the Data Analyst role and then shared with appropriate users within your organization. The Data API area can be used to retrieve project data and other data from the Accolade database, and display the Accolade information using your company's BI application.

Chapter 2

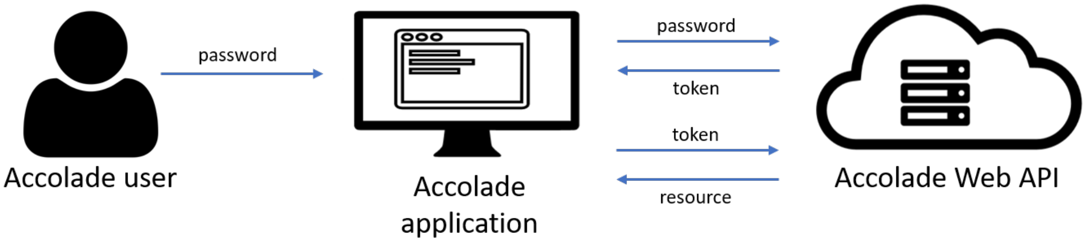
API Authentication as an Accolade User

Access to the Accolade Web API areas is available to any *in-network* authorized developer, and requests are authenticated via the user's Accolade credentials.

Authentication schemes are designed to serve two main purposes:

- User authentication - verify that the API user making the call has the appropriate credentials.
- User authorization - check whether the API user making the call has permission to make this kind of request.

Authentication schemes provide a secure way to identify the API caller. An endpoint can also check with the authentication token to confirm that permission has been granted for it to make a call to the API. Based on the information available on the authentication token, the API server determines whether to authorize that particular request.



Important! The Accolade Web API is sensitive to the user roles and rights that are assigned within Accolade. When creating API calls that will be made on behalf of an Accolade user, be mindful that the user must have the appropriate Accolade system roles and rights to access the data or perform the action included in the call. For example, Team Members do not have the appropriate rights to create projects, therefore it would not make sense to assign a POST call that creates projects for this user to execute.

Note: Roles and rights are not applicable to calls made to the Accolade Data API that are authenticated via a generated API key. See the API Authentication as an Application topic in the online Help for more information.

User Authentication Setup

Access to the Accolade Web API is available to any *in-network* authorized developer, and requests are authenticated via the user's Accolade credentials using the Bearer authentication method.

If the call is made from outside Accolade, you should use the `/Token` and/or the `/Account/Authenticate` endpoints to authenticate the user. If the client is a browser, `CORS` should be enabled on Accolade as well.

Requesting the Accolade Authentication Mode

To figure out the Accolade authentication mode:

Request	Response

POST https://<server>/Token
Content-Type: application/json
{
"grant_type": "auth_mode"
}
	200 OK
	Content-Type: application/json; charset=utf-8
	{
	"auth_mode": "Windows"
	}

Modes are: Windows and SSO.

Requesting an Access Token

To request a user-based access token:

Request	Response

POST https://<server>/Token
Content-Type: application/json
{
"grant_type": "password",
"username": "<login name>",
"password": <password>"
}
	200 OK
	Content-Type: application/json; charset=utf-8
	{
	"access_token": "<token>",
	"expires_in": "<seconds>",
	"refresh_token": "<token>"
	}

The access token expires in expires in seconds. For refresh token, see the Refresh Tokens section below.

This works for all authentication modes unless SSO is configured with 2-factor (2FA) or multi-factor authentication (MFA). Currently there is no way to determine if 2FA or MFA is enabled. See SSO with two-factor or multi-factor authentication for more information.

Refresh Tokens

If refresh tokens is enabled, you can use a refresh token to get a new access token without supplying the user’s credentials again. The refresh token expires after 365 days or if the

Accolade user is disabled or deleted.

To get a new access token from a refresh token:

Request	Response

POST https://<server>/Token
Content-Type: application/json
{
"grant_type": "refresh_token",
"refresh_token": "<refresh token>"
}
	200 OK
	Content-Type: application/json; charset=utf-8
	{
	"access_token": "<token>",
	"expires_in": "<seconds>",
	"refresh_token": "<token>"
	}

Note that the returned refresh token is a new token with a new expiration window.

The refresh token that is generated contains the following IP addresses:

1. The IP address of the remote client (the first IP address in the forwarded header is the real remote client IP address)
2. All IP addresses listed the Forwarded HTTP request header if any (standardized forwarded header)
3. All IP addresses listed the X-Forwarded-For HTTP request header if any (non-standardized forwarded header)
4. The loopback/localhost IPv4 and IPv6 IP addresses are excluded

The IP address list is stored in the refresh token as a claim with the name *'http://sopheon.com/claims/originipaddress'*. You can use JSON Web Tokens to see the *'http://sopheon.com/claims/originipaddress'* claim value of a refresh token. When redeeming the refresh token, the generated IP address list from the endpoint request, is compared with the list stored in the refresh token. At least one IP address matches. If not, redeeming the refresh token fails with an 'origin not allowed' message.

Implicit Grant Flow

With implicit grant flows, tokens (ID tokens or access tokens) are returned directly from the /account/authenticate endpoint instead of the /token endpoint, meaning access tokens are returned in the URL in the UI.

Request	Response
GET https://<server>/account/authenticate	200 OK

Request	Response

Content-Type: application/json	Content-Type: application/json;
	{
	"access_token": "<...>",
	"expires_in": 1200,
	"refresh_token": "<...>",
	"refresh_expires_in": 31536000,
	"token_type": "Bearer"
	}

GET https://<server>/account/authenticate?notifyScript=true
Content-Type: application/json
	200 OK
	Content-Type: text/html;
	<!DOCTYPE html><html><body><script>
	window.external.notify('
	{
	"access_token":"<...>",
	"expires_in":1200,
	"refresh_token":"<...>",
	"refresh_expires_in":31536000,
	"token_type":"Bearer"
	}
	');
	</script></body></html>

GET https://<server>/account/authenticate?redirectUrl=https://server/callback Content-Type: application/json	200 OK redirects to https://server/callback?token=<base64 encoded token>
GET https://<server>//account/authenticate?notifyScript=<true or false>&redirectUrl=<url> if redirectUrl is specified and absolute URL if URL is allowed (auth:AllowedClientURIs setting, defaults to *)	200 OK redirects to redirectUrl with token in query string: https://server/redirect?token=<base64 encoded JWT token>

Request	Response
Content-Type: application/json	

GET https://<server>//account/authenticate?notifyScript=<true or false>&redirectUrl=<url>
if notifyScript is true
Content-Type: application/json
	200 OK
	Content-Type: text/html;
	`<!DOCTYPE html><html><body><script>`
	`window.external.notify (<JWT token>);`
	`</script></body></html>`

GET https://<server>//account/authenticate?notifyScript=<true or false>&redirectUrl=<url>
Content-Type: application/json
	200 OK
	Content-Type: application/json;
	`<JWT token>`

Call the API with the Access Token

Each call to the API needs a HTTP Authorization header with Bearer scheme to authenticate the user:

Request	Response
GET https://<server>/api/v2/core/projects Authorization: Bearer <access token>	NA

SSO with two-factor or multi-factor authentication

Single sign-on authentication (or SSO authentication) allows users to log in once to access multiple applications, services. and accounts.

If the API client runs in a browser you should redirect to: `https://<server>/Account/Authenticate?redirectUrl=<url>`. This will show the SSO login dialog in the browser window. After successful login, the browser is redirected to the `redirectUrl` and the access token is returned as a query string. The query string is a base64 encoded JSON object `(?token=<base64 string>).`

If you want to cover all SSO configurations, with or without 2FA or MFA, don't use the `/Token` endpoint but always use `/Account/Authenticate` endpoint for the Federation authentication mode.

Configuration

Here are the settings that controls the authentication mode and token lifetimes.

Settings	Value
auth:AllowedClientURls	Allowed redirect URLs on `https://<server>/Account/Authenticate?redirectUrl=<url>` requests.
auth:AuthenticationMode	`Windows` or `Federation`. Defaults to `Windows`.
auth:SessionLifetime	Access token lifetime. Defaults to the Accolade `SessionTimeout` setting.
auth:EnableRefreshToken	If refresh token is be supported or not. Defaults to `True`.
auth:EnableWebRedirect	Whether or not the redirect to the identity provider should be done by a client-side redirect instead of returning a 302 Redirect response. Defaults to `True`.
auth:RefreshTokenLifetime	Refresh token lifetime. Defaults to `365 days`.

Notes:

- If the Web API is called from within Accolade, for example in the configuration of a quick grid, the user is already authenticated and no further authentication is needed.
- All API requests should be made over HTTPS. Calls made over plain HTTP could potentially expose information such as passwords or client secrets to other users within your network.
- The Accolade Web API technical documentation, which includes examples, can be accessed at *<server name>/help/apihelp*.
- In you are unfamiliar with working in APIs or require more technical support, contact Sopheon's Consulting group for more information on training opportunities or additional services.

Appendix A

Accolade Web API Developer References

This section contains developer information about the Accolade Web API resources.

Accolade Web API Design

The Accolade API is a private RESTful web service, intended for use by company assets and by our customers. Accolade's API conforms to the Richardson Maturity Model Level 2. It is implemented with ASP.NET Web API 2.2 with attribute routing, and follows OData v3 conventions.

Versioning

The Accolade Web API is semantic versioned, and the version number is independent from the Accolade version. The version number has the MAJOR.MINOR format and is incremented as follows:

- **MAJOR** version when there are **incompatible** API changes.
- **MINOR** version when functionality is added in a **backwards-compatible** manner.

Accolade v.17.1 currently supports Accolade Web API version 2.5.

Restrictions and Rate Limits

The only restrictions to the Accolade API are via authentication and authorization as follows:

- User authentication - verify that the API user making the call has the appropriate credentials.
- User authorization - a check whether the API user making the call has permission to make this kind of request.

Important! The Accolade Web API is sensitive to the user roles and rights that are assigned within Accolade. When creating API calls that will be made on behalf of an Accolade user, be mindful that the user must have the appropriate Accolade system roles and rights to access the data or perform the action included in the call. For example, Team Members do not have the appropriate rights to create projects, therefore it would not make sense to assign a POST call that creates projects for this user to execute.

Note: Roles and rights are not applicable to calls made to the Accolade Data API that are authenticated via a generated API key. See Accolade Web API Authentication as an Application for more information.

Rate limits determine how frequently you can call a particular endpoint. While Accolade's API does not currently impose any rate limits, be mindful that API usage may affect the performance of Accolade; you may wish to schedule large updates or requests outside of peak usage times.

Resources

A resource is uniquely identified by its URI (or URL). Resources URIs must follow the following conventions:

- Resources are grouped in **areas**. Examples: `core, administration, configuration`.
- Resources are **nouns** that convey meaning. Examples: `/api/v2/core/projects, /api/v2/configuration/processmodels`.
- Resource names should be **pluralized**. Examples: `/api/v2/core/projects, /api/v2/administration/users`.
- A single resource is addressed with its **ID in parenthesis** after the resource name. Examples: `/api/v2/core/projects(1), /api/v2/administration/users(42)`.
- Alternatively, a resource may be addressed with its **name or system name** in single quotes. Examples: `/api/v2/core/projects('Project%201'), /api/v2/configuration/metrics('Costs')`.
- Resources represent **contained structure**. Structure is represented by **hierarchical resource paths**. Examples: `/api/v2/core/projects(1)/metrics, /api/v2/core/projects(1)/metrics(111)`.
- Filters, sorting, etc. aren't resources thus should be in the **query string** of the URL. Example: `/api/v2/core/projects?$orderby=Name`.

Resource IDs should be unique, static and remain the same over time.

Verbs

The following verbs (**HTTP Methods**) are supported:

HTTP Method	Request Payload	Sample URI	Response Payload

GET	-	`/api/v2/core/projects`
		`/api/v2/core/projects({projectID})`
			Resource collection
			Single resource

POST	Single resource	`/api/v2/core/projects`	Single resource
PUT	Single resource	`/api/v2/core/projects({projectID})`	Single project

PATCH	Batch of partial resources
	Partial resource
		`/api/v2/core/projects`
		`/api/v2/core/projects({projectID})`
			Resource collection

HTTP Method	Request Payload	Sample URI	Response Payload
			Single resource
DELETE	-	`/api/v2/core/projects({projectID})`	-

💡 The difference between **PUT** and **PATCH** is that **PUT** will update the whole resource, so for not supplied properties their default values will be used. **PATCH** only updates supplied properties.

Status Codes

Status Code	HTTP Methods	Comments
Level 200 - Success		
200 - OK	GET, PUT, PATCH	And POST for Actions.
201 - Created	POST	
204 - No content	DELETE	
Level 400 - Client Errors		
400 - Bad request	POST, PUT, PATCH	Invalid or corrupt request payload, the response body contains the error message.
401 - Unauthorized	All	No or invalid authentication details provided.
403 - Forbidden	All	Authenticated user doesn't have access to resource.
404 - Not found	GET, PUT, PATCH, DELETE	Resource not found.
405 - Method not allowed	All	HTTP method not allowed on resource.

Status Code	HTTP Methods	Comments
406 - Not acceptable	All	Media type not supported (response payload).
409 - Conflict	All	Caching or concurrency conflict (or resource already exists when trying to create it).
415 - Unsupported media type	POST, PUT, PATCH	Media type not supported (request payload).
Level 500 - Server Faults		
500 - Internal server error	All	Error will be logged in the Accolade log. No error message will be returned as this exposes implementation details.

Payload

Resource != Business Model != Entity Model

Resources are mapped to one or more entities or their subsets. For example, a metric can contain both project and process model related properties.

A entity can be represented by more than 1 resource, depending on their usages. For example, a metric can have a separate resource type for: process model, project, matrix, planning/roadmap, etc.

All resources will have standardized property names and standardized value types as follows:

- **Enum** values are represented as **strings** and not as their underlying numeric value.
- **Dates** as **'YYYY-MM-DD'** strings.
- **Timestamps** as **'YYYY-MM-DDThh:mm:ss'** strings.

Related resources will be represented with their **ID** and can be expanded (with the `$expand` option) to full resource objects to reduce server round trips; for example, the team leader ID on a project can be expanded to a user resource.

Only JSON (media type: **application/json**) will be supported for request and response payloads. JSON literals will be in **camel-case**. Null values will be **omitted** from the payload to reduce network bandwidth.

Actions and Functions

Besides resources, actions and functions can be exposed by the API; this should be used sparingly.

The difference between actions and functions is that **actions** can have **side effects**, and **functions** do **not**. Both actions and functions can return data.

Actions and function should be **named** like C# method names, in other words, the name should be a **verb phrase**. **Actions** should be invoked with a **POST** method, **functions** with a **GET** method.

Actions and functions can be global, for a resource collection or for a single resource.

Examples:

```
api/v2/administration/users/SetPushSubscription
```

```
api/v2/core/ExecuteQuery(123)
```

```
api/v2/core/projects(42)/phases(1)/gate/keepers(5})/EnterVote
```

```
api/v2/core/projects(42)/MigrateProject(111)
```

OData

A request returns a maximum of 50 items in a collection, for example `api/v2/Projects` returns 50. To avoid the limitation you can use the OData $top token and set the results count to any number you need in the form:

```
api/v2/Projects?$top=100000
```

OData parameters are passed as a regular query string. For example `http://domain.com/page?$top=100&$skip=10` will pass 2 parameters, `$top` and `$skip`. You can pass as many parameters as you like, separated with a &.

Paging, Filtering, and Data Shaping

The following options are supported on resources collections (`$expand` and `$select` are also applicable on a single resource).

Option	Description	Format
$expand	Expands related entities inline.	$expand is a comma-separated list of related resources to be included in line with the retrieved resources. Nested resources can be expressed using a slash (‘/’), max depth is 10. **Example:** `/api/v2/core/projects(42)?$expand=Metrics,Phases/Stage/Deliverables`
$filter	Filters the results, based on a Boolean condition.	$filter is a Boolean expression to filter a collection of resources. The expression specified with the filter is evaluated for each resource and when the expression evaluates to true, the resource is included in the result. All OData v3 operators and functions are supported; see http://www.odata.org/documentation/odata-version-3-0/url-conventions/, section ’5.1.2. Filter System Query Option’. **Example:** `/api/v2/core/projects?$filter=Name+eq+'Ajax'` 💡 Nested $filter statements are not supported ($expand=Metrics($filter=DataType+eq+'String'))
$inlinecount	Include the total count of matching entities in the response.	$inlinecount with the value allpages returns the total number of resources in the (filtered) collection in the all-pages-count HTTP header. **Example:** `/api/v2/core/projects?$inlinecount=allpages`
$orderby	Sorts the results.	$orderby is a comma-separated list of property order clauses for resources to be returned in either ascending order using asc or descending order using desc. If asc or desc not specified, then the resources will be ordered in ascending order. **Example:** `/api/v2/core/projects?$orderby=TeamLeader+asc,Name+desc` 💡 Nested $orderby statements are not supported

Option	Description	Format
		💡 ($expand=Metrics($orderby=SystemName)).
$select	Selects which properties to include in the response.	$select is a comma-separated list of resource properties to be returned. Nested resources can be expressed using a slash ('/'), max depth is 10. **Example:** `/api/v2/core/projects(42)?$expand=Metrics&$select=Name,Code,Metrics/SystemName`
$skip	Skips the first in results.	$skip is a number that defines the number of resources to be skipped and not included from a (filtered) collection. **Example:** `/api/v2/core/projects?$skip=10`
$top	Returns only the first in the results.	$top is a number that defines the number of resources to be returned from a (filtered) collection. If not supplied, all resources in collection are returned. **Example:** `/api/v2/core/projects?$top=5`

💡 As of Accolade v. 13.2, support has been added for some DTO's to allow for filtering by expanded properties. Example: /api/v2/core/projects?$expand=TeamLeader&$filter=TeamLeader/Name+eq+%27FirstName%20LastName%27

Notes:

If a resource or action returns tabular data, the format query string can be used to shape the tabular data: `List,` `Table` and `Table with Headers:`

- `api/v2/core/ExecuteQuery`
- `api/v2/core/RunReport`
- `api/v2/core/projects/matrices`
- `api/v2/configuration/referencetables`

Alternative Keys

The APIv2 resource ID arguments can have 3 types of values:

- `Int32`: resource(123)
- `String`: resource("name")
- `Enum`: resource(Flag)

Caching

Each response will define itself a cacheable or not using HTTP Caching (RFC 2616 and RFC 7234 standards). Caching is used to eliminate the number of requests (aka network-roundtrips; uses an expiration model) and to eliminate the need to send full responses (aka network bandwidth; uses a validation model).

The following cache types will be supported in the future:

- Client Cache (== Private Cache), lives on the client
- Gateway Cache (== Shared Cache), lives on the server
- Proxy Cache (== Shared Cache), lives on the network

The expiration model will be defined with the Cache-Control header. The validation model will be defined with the `Last-Modified` and `ETag` (strong and/or weak) headers. Only `no-cache` is supported for now.

Notes:

`api/v2/core/ExecuteQuery` and `api/v2/core/RunReport` already support `Last-Modified` and `ETag` to avoid re-executing queries and reports when paging is used (`$top` and `$skip`).

Concurrency

APIv2 will follow current Accolade practices with respect to concurrency: last save wins.

Cross-Origin Resource Sharing (CORS)

JSONP support is dropped in APIv2. Only CORS will be supported. The existing Accolade CORS infrastructure will be used.

Security

APIv2 follows all Accolade visibility, accessibility and manageability rules for resources and their underlying entities (role, access groups, security list, security profiles, team membership, etc.).

If APIv2 is called from within Accolade (i.e. Quick Grids), the user is already authenticated and the session and authentication cookies will be send with each request automatically.

If APIv2 is called from outside Accolade and Accolade is configured for Windows Integration authentication, APIv2 can be called without using an access token (providing that the HTTP request stack supports this).

If APIv2 is called from outside Accolade and Accolade is configured for SSO (WS-Fed, SAMLp, OAuth2 and/or OpenID Connect), an access token should be requested. The access token should be supplied on each request as HTTP Authorization header with Bearer scheme (Authorization: Bearer <access token>).

Currently only the Resource Owner Password Credentials Grant flow is supported by Accolade. This flow requires the submission of username and password to obtain the access token (see /Token endpoint). Also, the OAuth2 offline scope (aka refresh tokens) is supported; this means a refresh token (that has a long expiration date) can be used to get an new access token without supplying the user credentials again.

Note: Currently there is no other means than deleting or de-activating the user in Accolade to revoke access and refresh tokens.

Batch Support

A batch request combines multiple APIv2 requests into a single POST request to the `/api/v2/$batch` endpoint. The payload should be multipart/mixed.

Request Example (only showing `GET` requests, but `POST, PUT, PATCH,` and `DELETE` requests can be included as well:

Request

```
POST http://<server>/api/v2/$batch HTTP/1.1
Content-Type: multipart/mixed; boundary="batch_e5b6e99a-61b3-4369-9331-c87803c7089a"
Host: <host>
Content-Length: 409
Expect: 100-continue
--batch_e5b6e99a-61b3-4369-9331-c87803c7089a
Content-Type: application/http; msgtype=request
GET /api/v2/administration/users(1) HTTP/1.1
Host: <host>

--batch_e5b6e99a-61b3-4369-9331-c87803c7089a
Content-Type: application/http; msgtype=request
GET /api/v2/core/projects HTTP/1.1
Host: <host>
```

Request

```

--batch_e5b6e99a-61b3-4369-9331-c87803c7089a--
```

Response Example (only showing `GET` requests, but `POST, PUT, PATCH`, and `DELETE` requests can be included as well:

Response

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 507
Content-Type: multipart/mixed; boundary="2d36ec9b-bf61-4ba9-
b265-ffb7604cff0a"
Expires: -1
Persistent-Auth: true
X-Frame-Options: SAMEORIGIN
Date: Wed, 31 May 2017 13:49:24 GMT

--2d36ec9b-bf61-4ba9-b265-ffb7604cff0a
Content-Type: application/http; msgtype=response

HTTP/1.1 200 OK
api-version: 2.0
Content-Type: application/json; charset=utf-8

{
"id": 1,
"name": "Administrator"
}
--2d36ec9b-bf61-4ba9-b265-ffb7604cff0a
Content-Type: application/http; msgtype=response

HTTP/1.1 200 OK
api-version: 2.0
Content-Type: application/json; charset=utf-8
```

Response

```
[
{
"code": "1",
"id": 1,
"name": "Ajax"
}
]
--2d36ec9b-bf61-4ba9-b265-ffb7604cff0a--
```

Standardized Properties

Name	Type	Description
Id	Long	Resource ID
Name	String	Resource display name
SystemName	String	Resource system name, unique across resource type, case insensitive
Description	String	Resource description
Order	Long	Position of resource in collection
action**Date**	Date	Date of action **Examples:** `CreatedDate ClosedDate` (no `LastModifiedDate` but `UpdatedDate` instead)
action**ById**	Long	ID of user that executed the action **Examples:** `CreatedById, ClosedById`
Is_Flag_	Boolean	A flag **Examples:** `IsClosed, IsActive, IsCalculated, IsShowMessagesEnabled`
Can_Flag_	Boolean	A flag **Examples:** `CanCreateStatusReports`
resource**Id**	Long	ID of related resources **Examples:** `TeamLeaderId, ClassId, ProcessModelId`
resource	DTO	Expanded related resource

Name	Type	Description
		Examples: `CreatedBy, ClosedBy, TeamLeader, Class, ProcessModel`
resource + **Parent***Resource*	DTO[] + DTO	Hierarchical relationships **Examples:** `/aNodes + ParentNode, Groups + ParentGroup`
Options	Enum	Options and flags
Members	DTO[]	Resource members **Examples:** `Project Team, AccessGroup, SecurityList`
ExtendedFields	DTO[]	Metadata and extended fields **Examples:** `Project Metadata, Deliverable ExtendedFields, User ExtendedFields`
Notes	String	**Examples:** `StatusText, Comments`
*type***DownloadUri**	String	**Examples:** `ProjectDocumentVersionDto.DownloadUri, GateDocumentDto.TemplateDownloadUri, ImageDto.DowloadUri`
ModelId	Long	Resource model ID **Examples:** `ProcessModelId, PMPhaseId, PMDeliverableId, PMActivityId, PMGateId`
Links	DTO[]	**Examples:** `Associations, ProjectLinks`
RequestorRights	Enum	Resource rights and permissions for the API caller.

Accolade Web API Navigation

In addition to the overview information provided within the online Help, Accolade provides technical documentation for developers that is available with your Accolade installation. The documentation, which includes the available methods and call examples, can be accessed at *<server name>/help/apihelp*.

If you are unfamiliar with working in APIs or require more technical support, contact Sopheon's Customer Support group for more information on training opportunities or additional services.

The Accolade Web API documentation is divided into areas representing different areas where information can be accessed within Accolade.

Category	Description
Administration	This category contains the calls that are specific to access and security within Accolade.
Configuration	This category contains the calls that are specific to the configuration components within Accolade.
Core	This category contains the calls that are specific to project data within Accolade.
Data	This category contains the calls that are specific to accessing the data tables for use with your company's BI tool of choice. **Important!** This category requires an Accolade API key in order to access, and should not be used with any advanced configuration within Accolade. See Creating and Managing Accolade Data API Keys for more information.
Miscellaneous	This category contains the calls that are not applicable to the remaining categories.
Resource Planning	This category contains the calls that are specific to the Resource Planning data within Accolade.

Sopheon Corporation

6870 West 52nd Avenue, Suite 215

Arvada, CO 80002