April 2021

Working with Absolute APIs

Our APIs provide you with access to Absolute functionality and data without having to use the Absolute console.

This document provides information about:

- accessing and authenticating APIs
- preparing requests for APIs
- authenticating headers in a request
- filtering and sorting
- troubleshooting

IMPORTANT Absolute APIs support connections using Transport Layer Security (TLS) protocol version 1.2 only.

Accessing and authenticating APIs

To access the Absolute APIs, you use the Absolute console to perform the initial setup, which includes assigning user roles and providing access to the console. You must then create and manage API tokens that are required for authentication and authorization.

API access

The URL you use to access the Absolute APIs depends on which URL you use to access the Absolute console:

- If you use https://cc.absolute.com to access the console, use the following URL to access the API:

  https://api.absolute.com
- If you use https://cc.us.absolute.com to access the console, use the following URL to access the API:

  https://api.us.absolute.com
- If you use https://cc.eu2.absolute.com to access the console, use the following URL to access the API:

  https://api.eu2.absolute.com

NOTE Examples in this document use https://api.absolute.com.

Prerequisites

There are two prerequisites before developers can use the Absolute APIs. The developer:

- must be assigned an Absolute user role. The role can be a default user role, or a custom user role defined by your organization.
- has at least one API token in the Absolute console.

API tokens

An API token consists of two parts: token ID and secret key. The token ID is a random UUID, and the secret key is generated with a crypto-level random number generator. The token has the equivalent level of permission as the assigned user role.

NOTE Ensure that the developer's user role is granted appropriate permissions. For example, to create a Freeze request using the Device Freeze API, the user role associated with the API token must be granted *Perform* permissions for Freeze Device.

NOTE API tokens don't expire. However, if the user account associated with the token is suspended or deleted, the token is no longer valid.

Token ID

The token ID is a random GUID-like string and is public information, like a user name. It is associated with the same role and device group as the Absolute user account.

Secret key

The secret key is a random sequence of bits and is private and sensitive information.

IMPORTANT Store this key securely, and do not share it.

Creating an API token

➔ **To create an API token containing a generated token ID and secret key:**

1. Log in to the Absolute console.
2. From the quick access toolbar, on any console page, click ⊕ **> API Token**.
3. On the API Token Management page, click **Create token**.

   The Create Token dialog appears.
4. Enter a **Token name** and **Description**.
5. Click **Save**.

   The Token Created dialog displays your generated token ID.
6. Download the token ID and secret key or view the secret key.

   WARNING If you close this dialog before downloading or copying the secret key, you cannot retrieve it later.

   - To download the token ID and secret key:
     a. Click **Download Token**.
     b. Save the .token file.
     c. Use a text editor to open and view the file.
   - To view the secret key:
     a. Click **View Secret Key**.

The **Secret key** is populated.

b. Copy both values of the **Token ID** and **Secret key** to a text file and then save the file.

7. Click **Close**.

On the API Token Management page, the new token is added to your list of tokens.

NOTE If a 401 error causes the API authentication to fail, you can enable authentication debugging from this page.

WARNING The secret key is comparable to a password. Keep it secure, and do not share it with anyone.

For more information about editing or deleting API tokens, see *Managing tokens used to access Absolute APIs* in the online help.

Preparing requests for APIs

You must use proper format and include your token in order to properly authorize your API request. To make an API request, you:

1. Create a canonical request
2. Create a signing string
3. Create a signing key
4. Create a signature
5. Add the authorization header

This document provides some basic examples. For more code samples, contact Absolute Technical Support (www.absolute.com/en/support).

Creating a canonical request

The canonical request looks like this:

```
CanonicalRequest =
  HTTPRequestMethod + '\n' +
  CanonicalURI + '\n' +
  CanonicalQueryString + '\n' +
  CanonicalHeaders +
  LowerCase(HexEncode(Hash(RequestPayload)))
```

The following table describes the parameters:

CanonicalRequest parameters

Parameter	Description
HTTPRequestMethod	All uppercase request methods such as GET, POST, and so forth.
CanonicalURI	The Request path, excluding the hostname and query parameters Path segments are URL-encoded in case they contain spaces or other characters. • If there is no path, use "/" • Normalize the path using URI generic syntax **Example 1** URL: https://api.absolute.com/v2/reporting/devices?$top=2 URI path: /v2/reporting/devices CanonicalURI: v2/reporting/devices **Example 2** URL: https://api.absolute.com/v2/complex path/with spaces?$select=foo URI path: /v2/complex path/with spaces CanonicalURI: /v2/complex%20path/with%20spaces
CanonicalQueryString	The entire query string, URL-encoded If no query-string is present, use an empty string "" • Create a list of all arguments • Sort arguments in ascending order; for example, 'A' is before 'a' • If not already encoded, URI encode the parameter name and value using URI generic syntax • Reassemble the list into a string For each argument in argumentList: { str += argument.name + '=' + argument.value; if ! last argument str += '&'; }

CanonicalHeaders	Only a subset of headers is included
	Example pseudocode
	`CanonicalHeaders=""; //For each header in ProtectedHeaders`
	`{ CanonicalHeaders+= lowercase(header)`
	`+ ':'`
	`+ trimmed(header value)`
	`+ '\n';`
	`}`

Encoded hash of payload	Hash the entire body using SHA-256 algorithm, HexEncode, and apply lowercase

Example of a basic canonical request

```
GET
/v2/reporting/devices

host:api.absolute.com
content-type:application/json
x-abs-date:20170926T172032Z
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

Example canonical request with one query parameter

```
GET
/v2/reporting/devices
%24filter=substringof%28%2760001%27%2C%20esn%29%20eq%20true
host:api.absolute.com
content-type:application/json
x-abs-date:20170926T172213Z
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

Example canonical request with two query parameters

```
GET
/v2/reporting/devices
%24filter=substringof%28%2760001%27%2C%20esn%29%20eq%20true%20and%20substringof%28%2760000%
27%2C%20esn%29%20eq%20false
host:api.absolute.com
content-type:application/json
x-abs-date:20170926T172255Z
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

Creating a signing string

The signing string uses this format:

```
StringToSign =
 Algorithm + \n +
 RequestDateTime + \n +
 CredentialScope + \n +
 HashedCanonicalRequest
```

Parameters

The following table shows descriptions and examples of the parameters of the signing string:

StringToSign parameters

Parameter	Description	Example
Algorithm	The string used to identify the algorithm	`ABS1-HMAC-SHA-256`
RequestedDateTime	The date and time (in UTC) from **X-Abs-Date** Format: ***<YYYY><MM><DD>***T***<HH><MM><SS>***Z	`20170926T172032Z`

Parameter	Description	Example
CredentialScope	The CredentialScope is defined in three parts: 1. the date (in UTC) of the request Format: ***YYYYMMDD*** 2. region or data center (*must* be in lowercase) Possible values: • cadc • usdc • eudc **NOTE** Each data center has a unique URL. 3. version or type of signature Always **abs1**	`20170926/cadc/abs1`
HashedCanonicalRequest	The hashed, hex-converted, and lowercase value of the canonical request.	`63f83d2c7139b6119d4954e6766ce90871e413 34c3f29b6d64201639d273fa19`

Example of a string to sign

```
ABS1-HMAC-SHA-256
20170926T172032Z
20170926/cadc/abs1
63f83d2c7139b6119d4954e6766ce90871e41334c3f29b6d64201639d273fa19
```

Creating a signing key

HMAC-SHA256 is used for authentication.

The following table shows descriptions of the inputs used to create a signing key:

Signing key inputs

Input	Description
kSecret = UTF8.GetBytes(“ABS1”+ secret) **Equivalent alternative pseudo code** `kSecret = UTF8.GetBytes(String.Concatenate(“ABS1”,secret))`	The kSecret value is calculated by concatenating the static string “ABS1” with the value of the secret key from your API token and then encoding the resulting string using UTF8. The secret is the secret key value from the token that you created in the Absolute console.
kDate = HMAC(kSecret, Date)	The date (in UTC) of the request Format: ***<YYYY><MM><DD>*** The result is a byte array
kSigning = HMAC(kDate , "abs1_request")	Use the binary hash to get a pure binary kSigning key **NOTE** Do not use a hexdigest method. The result is a byte array.

Creating a signature

As a result of creating a signing key, kSigning is used as the key for hashing. The StringToSign is the string data to be hashed.

The signature looks like this:

```
signature = lowercase(hexencode(HMAC(kSigning, StringToSign)))
```

Parameters

The following table shows describes the parameters.

Signature parameters

Parameter	Description
kSigning	The byte array that was created from the signing key
StringToSign	The signing string

Example of signing the string

This example shows the resulting signature for a request which is then used in the authorization header.

```
Signature=e15b64a4f91a0e53c2f91a6f52756a74bc21e6f175795cbf85bc15e8ef32aab5
```

Adding the authorization header

Use the standard HTTP Authorization header.

```
Authorization: <algorithm> Credential=<token id>/<CredentialScope>,
SignedHeaders=<SignedHeaders>, Signature=<signature>
```

Parameters

The following table shows descriptions and examples of the parameters of the authorization header.

Authorization header parameters

Parameter	Description	Example
Authorization	The string used to identify the algorithm	`ABS1-HMAC-SHA-256`
Credential	The token ID	`cc2423f2-cc28-48a6-9dce-a268d5e3cd01`

Parameter	Description	Example
CredentialScope	The CredentialScope is defined in three parts: 1. The date (in UTC)of the request Format: ***<YYYY><MM><DD>*** 2. Region or data center (*must* be in lowercase) Possible values: • **cadc** • **usdc** • **eudc** **NOTE** Each data center has a unique URL. 3. Version or type of signature	`20170926/cadc/abs1`
SignedHeaders	Semi-colon (;) delimited list of lowercase headers used in CanonicalHeaders	`host;content-type;x-abs-date`
Signature	The fully calculated resulting signature from the signing key and the signature	`e15b64a4f91a0e53c2f91a6f52756a74bc21e6f17 5795cbf85bc15e8ef32aab5`

Example authorization header

```
Authorization: ABS1-HMAC-SHA-256 Credential=cc2423f2-cc28-48a6-9dce-
a268d5e3cd01/20170926/cadc/abs1, SignedHeaders=host;content-type;x-abs-date,
Signature=e15b64a4f91a0e53c2f91a6f52756a74bc21e6f175795cbf85bc15e8ef32aab5
```

NOTE There is a space after each comma in the authorization header. It may not appear if you use copy and paste.

Authenticating headers in a request

We include only a small subset of HTTP headers in a request to minimize the possibility of proxies modifying them in transit.

You must use the following headers for all of your API requests.

Authentication headers

Header	Description	Example
Host	The domain name of the server where the request is sent	`api.absolute.com`
Content-Type	The media type of the resource	`application/json`

Header	Description	Example
X-Abs-Date	The automatically generated header that indicates the time (in UTC) the request was made Format: ***<YYYY><MM><DD>*** T***<HH><MM><SS>***Z.	`20170926T172032Z`
Authorization	The HTTP authorization header Format: ***<algorithm>*** Credential=***<token id>/CredentialScope***, SignedHeaders= ***<SignedHeaders>*** ,Signature= ***<signature>***	ABS1-HMAC-SHA-256 Credential=cc2423f2-cc28-48a6-9dce-a268d5e3cd01/20170926/cadc/abs1, SignedHeaders=host;content-type;x-abs-date, Signature=e15b64a4f91a0e53c2f91a6f52756a74bc21e6f175795cbf85bc15e8ef32aab5

Filtering and sorting

Absolute uses a subset of query options from Open Data Protocol (OData) for filtering and sorting. Odata version 1 and 2 are supported. OData query parameters must be alphabetized and URI encoded.

For more information about OData, see: https://www.odata.org/documentation.

The applicable OData system query options are:

- $filter
- $orderby
- $select
- $skip
- $top

$filter

The $filter system query option filters items included in the response with the specified expression.

Operators supported by $filter are described in the following tables:

Logical operators supported by $filter

Logical operator	Description	Example
Eq	Equal	To view a list of all devices with an Active status (agentStatus eq A): `GET /v2/reporting/devices?%24filter=agentStatus%20eq%20'A'`

Logical operator	Description	Example
Ne	Not equal	To view a list of all devices with a status that isn't Active status (agentStatus ne A): `GET /v2/reporting/devices?$filter=agentStatus%20ne%20'A'`
Gt	Greater than	To view a list of all devices with greather than 1 GB (1024^3 bytes) of available physical ram (availablePhysicalRamBytes gt 1073741824) `GET /v2/reporting/devices?%24filter=availablePhysicalRamBytes%20gt%201073741824`
Ge	Greater than or equal	To view a list of all devices with greater than or equal to 1 GB (1024^3 bytes) of available physical ram (availablePhysicalRamBytes ge 1073741824) `GET /v2/reporting/devices?%24filter=availablePhysicalRamBytes%20ge%201073741824`
Lt	Less than	To view a list of all devices with less than 1 GB (1024^3 bytes) of available physical ram (availablePhysicalRamBytes lt 1073741824) `GET /v2/reporting/devices?%24filter=availablePhysicalRamBytes%20lt%201073741824`
Le	Less thank or equal	To view a list of all devices with less than or equal to 1 GB (1024^3 bytes) of available physical ram (availablePhysicalRamBytes le 1073741824) `GET /v2/reporting/devices?%24filter=availablePhysicalRamBytes%20lt%201073741824`
And	Logical and	To view a list of all devices with less than 1 GB (1024^3 bytes) and more than 500 MB (500x1024^3 bytes) of available physical ram (availablePhysicalRamBytes lt 1073741824 and availablePhysicalRamBytes gt 524288000) `GET /v2/reporting/devices?%24filter=availablePhysicalRamBytes%20lt%201073741824%20and%20availablePhysicalRamBytes%20gt%20524288000`
Or	Logical or	To view a list of all devices with less than 1 GB (1024^3 bytes) or less than 1 GB (1024^3 bytes) of available virtual ram (availablePhysicalMemroyBytes lt 1073741824 or availableVirtualMemoryBytes lt 1073741824) `GET /v2/reporting/devices?%24filter=availablePhysicalRamBytes%20lt%201073741824%20or%20availableVirtualMemoryBytes%20lt%201073741824` /Products?$filter=Price le 3.5 or Price gt 200
Not	Logical negation	To view a list of devices a username that doesn't start with the domain MYCOMPANY (not startswith (domain,'MYCOMPANY') GET /v2/reporting/devices?%24filter=not%20startswith%28domain%2C%MYCOMPANY%27%29

Grouping operator supported by $filters

Grouping operator	Description	Example
( )	Precedence grouping	/Products?$filter=(Price sub 5) gt 10

Functions can also be used with $filter.

NOTE You can use a NULL literal in comparisons as ISNULL or COALESCE operators are not defined.

String functions supported by $filter

String function	Description	Example
substringof(string p0, string p1)	Returns a substring of the first parameter string value, starting at the Nth character and finishing at the second parameter integer value	http://host/service/Customers? $filter =substring(CompanyName, 1) eq 'bsolute'
endswith(string p0, string p1)	If the first string ends with the second string, it returns true; otherwise, it returns false	http://host/service/Customers?$filter =endswith(CompanyName,'Absolute')

$orderby

The $orderby system query option sorts a resulting list of data as you instruct. For example, you may want to sort the Device Report so that devices with the most current lastUpdatedUtc dates show first in the list (lastUpdatedUtc desc):

```
GET /v2/reporting/devices?%24orderby=lastUpdatedUtc%20desc
```

$select

The $select system query option requests a specific set of properties. For example, you may want to see only the manufacturer, model, and serial number attributes of your devices ($select=systemManufacturer,systemModel,serial):

```
GET /v2/reporting/
devices?%24select=systemManufacturer%2CsystemModel%2Cserial
```

$skip

The $skip system query option requests the number of items to be excluded from the result. For example, you may want the second page of results when data is returned in batches of 20 (skip=20&$top=20):

```
GET /v2/reporting/devices?%24skip=20&%24top=20
```

NOTE Use with the $top query option to paginate your results.

$top

The $top system query option requests the number of items to be included in the result. For example, you may want to limit the number of records returned to the first 10 (use $top=10):

```
GET /v2/reporting/devices?%24top=10
```

Troubleshooting

The following table lists some HTTP status codes and describes the corresponding errors that you may encounter when using Absolute APIs.

HTTP status code errors

Status code	Possible error	Mitigation
401	Incorrect HTTP method	Ensure that the method used in the calculation matches the actual request method.
	Incorrect HTTP method case	In the calculation, the HTTP method must be all uppercase; for example, GET, POST, PUT.
	Incorrect X-Abs-Date value or format	Ensure that the date value is accurate, is in the UTC timezone, and is in the ***<YYYY><MM><DD>***T***<HH><MM><SS>***Z format.
	Incorrect Query Parameters encoding	Ensure that the query parameters are URL-encoded when creating the CanonicalRequest. For example, the query parameter `$top=15` would be encoded as `%24top=15`.
403	The tokenID belongs to a user that does not have permission to access the URL.	Ensure that the user who created the token has the appropriate permissions.

If you are unable to resolve the errors listed above, enable **Authentication Debugging** in the Absolute console on the API Token Management page. After that, repeat the failed API requests to log the details. You can then contact Technical Support with the details collected in the log. Ensure to include the following:

- tokenID
- canonicalRequest
- x-abs-date
- signature

Copyright Information