Digital Vault

The Vault is a digital space where your customers store stuff — documents, contracts, statements, agreements, etc. They can then search for individual items based on keywords or attributes and retrieve them. In Vault-land, your customer is called an owner, The things they store in their Vault are called assets:

- Owners own assets.
- You are the third-party source of assets.

How do you identify an owner? With an ownerId. How do you identify an asset? With an assetId (which, by the way, also identifies an owner; more on that later). You push the assets to the Vault for owners and provide metadata for them to search through their assets. Assets are part raw data, part metadata.

From your customer's perspective, using the Vault is a bit like using a parking garage. First, the customer provides their name to reserve a parking assignment and receive a ticket. It's the customer's spot even though they haven't parked their car, yet. The customer can choose to describe their car (color, make, model, etc.) so the parking valet (you) can easily retrieve it in case they lose their ticket. The customer then arrives with their car and asks you to park it.

In this metaphor, the parking spot is an *asset* (identified by assetId), the customer is an *owner* (ownerId), and the car is the data that they want to store.

Getting it all into the Vault:

The steps to pushing an asset into a Vault are:

1. Identify an owner and get an ownerId
2. Reserve a space in the vault for the owner and get an assetId
3. Upload metadata that describes the "stuff" that the owner is going to push into the Vault
4. Push the actual data — documents, videos, sound files — into the Vault

The nice part? You can do steps 1-3 through a single call (Reserve a space in the Vault). For step 4 you need to call Push asset data into the Vault. Two calls, and your car is parked!

Other operations let you search for and retrieve assets, share assets with other customers, customize access permissions, and more.

Multiple owners of the same asset

Let's return to our parking garage metaphor and consider that a car may have more than one owner — the original customer and our customer's spouse, for example. This is where our metaphor breaks down. Each owner of an asset has their own assetId (their own parking ticket…that's okay) *and* their own copy of the asset's data (their own car…metaphor broken). For example, if one owner signs a co-signed document, the other owner won't see the signature on their copy of the document, at least not automatically.

Why does each owner have their own copy of the asset? Flexibility, primarily: Some vendors might not want every owner to have the same picture of the asset.

Operations

- [Find or create an owner]
- [Check owner status]
- [Reserve a space in the Vault]
- [Push asset metadata into the Vault]
- [Push asset data into the Vault]

Important	The following operations aren't currently supported. Although you can *push* data into the Vault, *retrieving* data requires a permission framework that's still being implemented.

- [Retrieve asset metadata from the Vault]
- [Retrieve asset data from the Vault]
- [Search for assets in the Vault]
- [Grant permissions on assets]
- [Revoke permissions on assets]

Find or create an owner

Every asset that's pushed into the Capital One Vault must have at least one owner. The owners are the end users to whom the asset applies: An account holder or the co-signers on a loan, as examples. Each owner in the Vault system is identified by a unique ownerId value.

It's this operation's responsibility to retrieve ownerId values for you: It returns an ownerId value that represents the person that you describe in the request body. If the ownerId value doesn't already exist, the operation creates it. If you describe the same person in subsequent calls to this operation, you'll receive the same ownerId value. In other words, each real-life person is represented by a single ownerId value.

To identify an owner, you supply as many attributes about the owner as you can; these attributes include name, email address, physical address, phone number, and so on. The more information that you supply, the more likely you are to find the correct real-world person (or, more accurately, to find the same person if you call this operation more than once).

In addition to the owner's personal information, you can supply a ownerIdInSourceSystem identifier. This is the customer identifier value that you use to identify the owner in your own customer-identification system. While not required, the ownerIdInSourceSystem value can help with auditing and trouble-shooting.

After you've retrieved one or more ownerId values, you use them in the Reserve a space in the Vault operation to assign owners to a Vault asset. Actually, you can use the Reserve a space in the Vault operation to identify owners and assign them to assets at the same time — you don't have to create the owners' ownerId values in this operation.

When an ownerId value is created, the Vault runs a verification process that determines if the customer can be used as a Vault asset owner. To check the status of this verification, call the Check owner status operation. You

may only use fully verified customers as asset owners.

`POST /vault/owner/match`

Request

Headers

Authorization* **string**

The Bearer token that authenticates your app with Capital One. Instructions for constructing the token are given in Authorization.

Content-Type* **string**

Specifies the MIME type of the data in the request body. The setting should be exactly this:

```
Content-Type: application/json
```

Accept* **string**

Specifies the MIME type that you'll accept in the body of the response *and* declares the version of the endpoint that you're invoking. The setting should be exactly this:

```
Accept: application/json;v=0
```

Body Properties

ownerIdInSourceSystem **string**

An optional, arbitrary string value that identifies the owner in your "native" customer-identification system. See the operation description, above for more details.

ownerDetails **object** ***default: none***

The properties in the ownerDetails object identify the person that you want to find.

> phoneNumber* **string** ***default: none***
>
> The owner's 10-digit phone number, hold the punctuation. For example:
>
> ```
> "phoneNumber": "8005551212"
> ```
>
> email* **string** ***default: none***
>
> The owner's email address. The form of the email address must be valid

("jsmith@example.com", for example); the Vault doesn't test the email to determine if it actually exists.

dateOfBirth **string** ***default: none***

The owner's date of birth given in ISO 8601 format, date only ("YYYY-MM-DD").

individual* **object** ***default: none***

The properties and subproperties in the individual object specify the owner's name.

firstName* **string**
lastName* **string**

The owner's name, broken into first and last. The minimum/maximum lengths and legal characters for these properties are listed below. As a convenience, the final column shows the regexp expression that you can use for validations:

property	length	legal chars	regexp
firstName	[1, 35]	Case-insensitive alphabet, hyphen, apostrophe, whitespace	`^[A-Za-z '-]{1,35}$`
lastName	[2, 35]	Case-insensitive alphabet, hyphen, apostrophe, whitespace	`^[A-Za-z '-]{2,35}$`

address **object** ***default: none***

The owner's mailing address.

streetAddress* **string**

city* **string**

stateCode* **string**

postalCode* **string**

countryCode **string** ***default: none***

The customer's mailing address:

property	meaning	format
streetAddress	Street address	Any characters, maximum 60
city	City name	Any characters, maximum 35

stateCode	State code	Two-character USPS code (US only). 50 states, plus: • **DC** (Washington DC) • **AA** (Armed forces, America) • **AE** (Armed forces, Europe) • **AP** (Armed forces, Pacific)
postalCode	Zip/postal code	Five-character USPS zip code (e.g: "5O310") or three+three character alphanumeric Canadian postal code (e.g: "K1A 0B1"). Canadian postal codes must include the space character.
countryCode	Country code	Three-letter country code (ISO 3166 alpha-3).

JSON Structure

```
{
  "ownerIdInSourceSystem": "be229b15966217327dfd2",
  "ownerDetails": {
    "phoneNumber": "8005551212",
    "email": "rwh@raywyliehubbard.com",
    "dateOfBirth": "1946-11-13",
    "individual": {
      "firstName": "Ray"
    },
    "address": {
      "streetAddress": "1230 Duck Fuss Lane",
      "city": "Beaumont",
      "stateCode": "TX",
      "postalCode": "77701",
      "countryCode": "USA"
    }
  }
}
```


Response

HTTP Status Codes

If the call returns a 4xx or 5xx HTTP status, the response body will contain properties that provide more information about the error.

200	The owner was successfully identified and an ownerId value, for that owner, is returned in the request body.

404	The client app or the owner identified by the ownerId query parameter doesn't have permission to call this operation, or one or more of the values in the assetId query parameter is invalid. (For security reasons, the no-permission case and the invalid assetId case are purposefully indistinguishable.)
429	The request has been rejected because of rate limiting -- you've sent too many requests in a given amount of time.
500	Server error.
502	Internal connection failure.
503	The server is unavailable due to heavy traffic or maintenance.

Body Properties

ownerId **string**

An opaque string value that uniquely identifies the owner that you specified in the request.

JSON Structure

```
{
  "ownerId": "12768723683746"
}
```

```
var https = require('https');
var queryString = require('querystring');

// Assemble the request message headers
var requestHeaders = {
  'Authorization': 'Bearer 9c776abc4a9633d625173',
  'Content-Type': 'application/json',
  'Accept': 'application/json;v=0'
}

// Write the request body
requestBody = JSON.stringify({
  "ownerIdInSourceSystem": "be229b15966217327dfd2",
  "ownerDetails": {
    "phoneNumber": "8005551212",
    "email": "rwh@raywyliehubbard.com",
    "dateOfBirth": "1946-11-13",
    "individual": {
      "firstName": "Ray"
    },
    "address": {
      "streetAddress": "1230 Duck Fuss Lane",
      "city": "Beaumont",
      "stateCode": "TX",
      "postalCode": "77701",
      "countryCode": "USA"
    }
  }
})

// Assemble the calling options for the request message
```

```
var options = {
  method: 'POST',
  hostname: 'api-sandbox.capitalone.com',
  port: 443, // https
  path: '/vault/owner/match',
  headers: requestHeaders
}

// Create the request and handle the response
var findOrCreateAnOwner = https.request(options, function(response) {

  // Accumulate the response data
  var responseData = "";
  response.on('data', function(data) {
    responseData += data;
  });

  // Process the response data
  response.on('end', function() {
    // Do something with responseData
  });
});

// Write the request body into the request message
findOrCreateAnOwner.write(requestBody);

// Finish sending the request
findOrCreateAnOwner.end();
```

```
-- REQUEST --
curl -i -k --tlsv1 -H "Authorization: Bearer 9c776abc4a9633d625173" -H
"Content-Type: application/json" -H "Accept: application/json;v=0" -d
'{"ownerIdInSourceSystem":"be229b15966217327dfd2","ownerDetails":
{"phoneNumber":"8005551212","email":"rwh@raywyliehubbard.com","dateOfBirt
h":"1946-11-13","individual":{"firstName":"Ray"},"address":
{"streetAddress":"1230 Duck Fuss
Lane","city":"Beaumont","stateCode":"TX","postalCode":"77701","countryCod
e":"USA"}}}' -X POST "https://api-
sandbox.capitalone.com/vault/owner/match"

--RESPONSE--
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 30
Date: Tue, 31 Oct 2017 12:54:09 GMT

{"ownerId":"12768723683746"}
```


Check owner status

When an ownerId is created (whether through Find or create an owner or Reserve a space in the Vault), the Vault determines if the connection between the potential owner and the vendor (i.e. you or the client for whom you're creating a Vault app) is valid and desirable to both the real-life customer and Capital One. This operation lets you retrieve the "connection status" for one or more owners.

The owners whose statuses you want to check are passed as a comma-separated list of ownerId values in the ownerId query parameter. For each ownerId that you pass in, you get an element in the owners array that's passed back in the response. Each element contains an ownerId and a connectionStatus property that will bear one of these values:

- Verified: The customer has been successfully identified and verified.
- Pending: The customer's identify is still being processed.
- Blocked: The customer's identity has been verified, but Capital One has blocked the connection, presumably for security reasons.
- Rejected: The customer has been identified, but has rejected the request to connect to the Vault.

While you can push data into the Vault for any owner regardless of their connection status, you may only *retrieve* asset data for owners who are Verified.

```
GET /vault/owner/status
```

Request

Query Parameters

ownerId* **array**

Comma-separated list of ownerId values whose connection statuses you want to check. All of the ownerId values must be valid; if any one of them isn't, the entire operation fails and returns a 404 status.

Headers

Authorization* **string**

The Bearer token that authenticates your app with Capital One. Instructions for constructing the token are given in [%Authorization</ a>.

Accept* **string**

Specifies the MIME type that you'll accept in the body of the response and declares the version of the endpoint that you're invoking. The setting should be exactly this:

```
Accept: application/json;v=1
```

Response

HTTP Status Codes

If the call returns a 4xx or 5xx HTTP status, the response body will contain properties that provide more information about the error.

200	Connection status values have been retrieved for all of the requested owners.
404	One or more of the values that you passed in the ownerId query parameter is invalid.
429	The request has been rejected because of rate limiting -- you've sent too many requests in a given amount of time.
500	Server error.
502	Internal connection failure.
503	The server is unavailable due to heavy traffic or maintenance.

Body Properties

owners **[object]**

Ann array of owner objects, one for each ownerId that you passed in the request's query parameter.

ownerId **string**

The owner's ownerId value.

connectionStatus **string**

The owner's connection status, one of:.

- Verified: The customer has been successfully identified and verified, and may be used as a Vault asset owner.
- Pending: The customer's identify is still being processed.
- Blocked: The customer's identity has been verified, but Capital One has blocked the connection, presumably for security reasons.
- Rejected: The customer has been identified, but has rejected the request to connect to the Vault.

While you can push data into the Vault for any owner regardless of their connection status, you may only *retrieve* asset data for owners who are Verified.

JSON Structure

```
{
  "owners": [
    {
      "ownerId": "a34e-ff87-6230-4875",
      "connectionStatus": "Verified"
    }
  ]
}
```

```
var https = require('https');
var queryString = require('querystring');

// Assemble the request message headers
var requestHeaders = {
  'Authorization': 'Bearer sd089jvws=0sa9nsiu$Uf',
  'Accept': 'application/json;v=1'
}

// Assemble the query parameters
var queryParams = {
  'ownerId': 'a34e-ff87-6230-4875,%20d87f-a345-cf9-e17a,%20as3f4-d827-
b333-4c4d'
}

// Assemble the calling options for the request message
var options = {
  method: 'GET',
  hostname: 'api-sandbox.capitalone.com',
  port: 443, // https
  path: '/vault/owner/status' + queryString.stringify(queryParams),
  headers: requestHeaders
}

// Create the request and handle the response
var checkOwnerStatus = https.request(options, function(response) {

  // Accumulate the response data
  var responseData = "";
  response.on('data', function(data) {
    responseData += data;
  });

  // Process the response data
  response.on('end', function() {
    // Do something with responseData
  });
});

// Finish sending the request
checkOwnerStatus.end();
```

```
-- REQUEST --
curl -i -k --tlsv1 -H "Authorization: Bearer sd089jvws=0sa9nsiu$Uf" -H
"Accept: application/json;v=1" -X GET "https://api-
sandbox.capitalone.com/vault/owner/status?ownerId=a34e-ff87-6230-
4875,%20d87f-a345-cf9-e17a,%20as3f4-d827-b333-4c4d"

--RESPONSE--
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 78
Date: Tue, 31 Oct 2017 12:54:09 GMT

{"owners":[{"ownerId":"a34e-ff87-6230-
4875","connectionStatus":"Verified"}]}
```

Reserve a space in the Vault

This operation's most important function is to reserve a space for an asset in the Vault. It also lets you provide metadata about the asset. After you've reserved the space — and possibly defined the metadata — you can then push the asset's data into the space through the Push asset data into the Vault operation.

To reserve a space for an asset, you must supply two things:

- A UI-appropriate name for the asset.
- A list of the asset's *owners*. These are the customers to whom the asset applies — the co-signers on a loan, or the depositor of a downpayment, as examples. An owner is identified through real-world attributes (name, email address, and so on) or by an ownerId value that you previously retrieved through the Find or create an owner operation. See the description of the owners property in the request body, below, for details.

After you've reserved the space, you then push the asset's metadata and data into it:

- You can supply the metadata in this call or you can provide it, later, by calling the Push asset metadata into the Vault operation.
- You push the asset's data through the Push asset data into the Vault operation, only.

Assets and Owners

When you reserve a space in the Vault for an asset and assign owners to that asset, you might expect that each of the owners has access to the same asset (i.e., the same underlying metadata and data). But that's not how it works. Every owner has their own copy of the asset, both the metadata and the data. This is important enough to repeat, with emphasis:

> **Every owner of an asset has their own copy of that asset!**

An owner's copy of the asset is identified by assetId; the assetId values that are created for an asset's owners are returned in the assets array in this operation's response.

If you have an asset (or, more accurately, an "asset space") that's shared by multiple owners *and* you want to keep the underlying data (including the metadata) in sync, it's your responsibility to make sure that when you push new data into the asset space, that you push it for each owner of the asset. How you do this depends on whether you're pushing the metadata or the data:

- The Push asset metadata into the Vault operation pushes metadata based on assetId. Since each owner has his or her own assetId, this means that you have to call the operation once for each owner.
- The Push asset data into the Vault is a bit easier: It accepts a list of assetId values through a query parameter. Thus, you can push the same data for all owners at the same time.

So why does each owner have his or her own copy of the asset? Flexibility, primarily: Some vendors might not want every owner to have the same picture of the asset.

New owners? New asset space…or just grant access permission

As mentioned above, when you reserve a space in the Vault for an asset, you have to identify the owners of that asset. And, as also mentioned, you can provide some metadata, as well. Owner identification is much more important than metadata definition. Why? Because you can change the metadata data later (through Push asset metadata into the Vault), but you *can't* change the set of owners. This is also worth stressing:

> * The only chance you have to identify an asset's owners is when you reserve a space for the asset.

But let's not get obsessed with an asset's owners. You can grant *access permission* to other individuals at any time through the Grant permissions on assets operation. In most cases, granting access permission is good enough, and it spares you the task of having to identify all of the interested parties before you reserve the asset space (i.e., before you call *this* operation).

And, just to be clear: The individuals to whom you grant access don't become owners of the asset.

`POST /vault/assets`

Request

Headers

Authorization* **string**

The Bearer token that authenticates your app with Capital One. Instructions for constructing the token are given in Authorization.

Content-Type* **string**

Specifies the MIME type of the data in the request body. The setting should be exactly this:

```
Content-Type: application/json
```

Accept* **string**

Specifies the MIME type that you'll accept in the body of the response *and* declares the version of the endpoint that you're invoking. The setting should be exactly this:

```
Accept: application/json;v=0
```

Body Properties

The properties in the request body fall into two categories:

- The owners property identifies the asset's owners.

- The other properties define the asset's metadata. Except for assetName, these are all optional properties; you can supply the metadata later through a call to Push asset metadata into the Vault.

owners* **[object]**

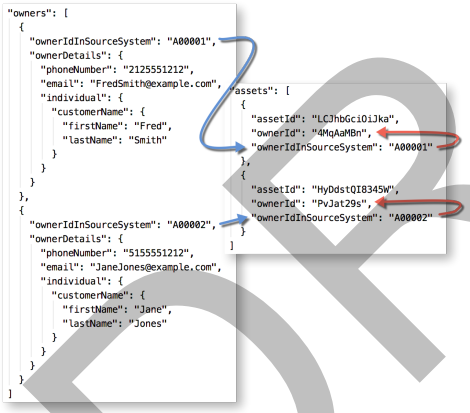
An array of objects that identify the asset's owners. These are people that have access to or otherwise are involved with the asset. To identify an owner you must supply either…

- …an "owner ID" through the ownerId property, or…
- …a description of the owner, using real-world attributes, in the ownerDetails object.

So where do you get an ownerId value? You can retrieve it through the Find or create an owner operation, or you can supply real-world information about the owner in *this* operation (the second approach, above) and the ownerId will be returned in the response. In either method, the procedure is the same: You provide real-world info, the operation gives you an ownerId that you can use in subsequent calls.

But that raises a question: Since this operation lets you identify multiple owners in a single call, how do you associate the various ownerId values with the real-world users to which they correspond? The answer: Through the ownerIdInSourceSystem property.

In the request, you set the ownerIdInSourceSystem property to a value that identifies the user in your native "user identification system". In the response, the operation returns an assets array that contains an object for each owner; each of these objects contains the ownerId value that the system generated as well as the ownerIdInSourceSystem value that you supplied in the request:



As shown above, the ownerIdInSourceSystem value that you supply for Fred Smith (`A00001`) lets you identify the reponse element that contains Fred's ownerId value (`4MqAaMBn`). Similarly for Jane Jones.

ownerId **string** ***default: none***

A Vault-generated unique identifier for this owner. See the owners property, above, for details.

ownerIdInSourceSystem **string** ***default: none***

ownerIdInSourceSystem is an arbitrary string value that identifies the owner in the caller's source system. See the description of the owners property, above, for details on how to construct and use the ownerIdInSourceSystem value.

> **Important** If you identify the owner through the ownerDetails object, the ownerIdInSourceSystem value is required.

ownerDetails **object** ***default: none***

This is the object that you use if you're identifying an owner through real-world information.

> **Note** For description of the ownerDetails object and its properties, see the ownerDetails property in the request message of Find or create an owner.

phoneNumber **string** ***default: none***

email **string** ***default: none***

individual **object** ***default: none***
- firstName **string**
- lastName **string**

address **object** ***default: none***
- streetAddress* **string**
- city* **string**
- stateCode* **string**
- postalCode* **string**
- countryCode **string** ***default: none***

assetName* **string**

A plaintext, human-readable, 255-character (maximum) name for the asset. The name, which needn't be unique, is used as fodder in the Search for assets in the Vault operation, so the longer and more descriptive the better.

assetType **string** ***default: none***

The format of the asset's data given as a MIME type such as `application/pdf`, `image/jpg`, or `text/csv`. For a complete list of MIME types, see https://www.iana.org/assignments/media-types/media-types.xhtml.

> **Note** Technically, the assetType value needn't be a MIME type. You can actually pass in any arbitrary string — the operation won't balk. Nonetheless, you're encouraged to stick to MIME types. To describe the asset's type by some other vocabulary, you should add a property of your own invention to the attributes object.

path **string** ***default: none***

The Vault directory into which the asset's data will be placed. This is an organizational convenience

that you can use to categorize your customers' assets, and that can be used when searching for an asset (through Search for assets in the Vault). For example:

```
"path": "/new_car/purchase/docs"
```

The rules for constructing a path value are:

- Alphanumeric characters, period, underbar, and hyphen only (regex: `[A-Za-z0-9._-]`)
- 4000 characters, max.
- Absolute paths, only: The path value must start with a forward slash (but needn't end in one).

tags **[string]** ***default: none***

A list of search terms that's consulted by the *Find an asset* operation. Each tag is a plaintext, 155-character (maximum) word or phrase. Tags are intended to be user-friendly: They're the sort of generically-descriptive terms that an end user would type into a search box.

attributes **[object]**

The attributes object comprises a set of properties that contain "administrative" information about an asset. This is information that's valuable in categorizing the asset for you, the vendor, but which isn't necessarily visible or even comprehensible to your customer.

There are no pre-defined properties in the attributes object. You get to define the object's properties by supplying both the name and the value. For example, here we define three properties:

```
"attributes" = {
  "loanType": ["auto"],
  "loanOfficers": ["Gabby Hayes", "Packer McGee"],
  "securityLevel": ["proprietary"]
}
```

- A property's name may contain alphanumeric characters, periods, hyphens, and underbars, only (regex: `[A-Za-z0-9._-]`).
- A property's value must be an array, even if it holds but a single element. The strings in the array must be UTF-8.

Attributes are searchable: They're used by the Search for assets in the Vault operation to locate and return assets that have a property with a given name and value.

JSON Structure

```
{
  "owners": [
    {
      "ownerId": "a34e-ff87-6230-4875",
      "ownerIdInSourceSystem": "M101",
      "ownerDetails": {
        "phoneNumber": "8005551212",
        "email": "string",
        "individual": {
```

```
        },
        "address": {
        }
      }
    }
  ],
  "assetName": "Judy and Sam new car loan; bank statements, April-June
2017",
  "assetType": "application/pdf",
  "path": "/new_car/purchase/docs",
  "tags": [
    "2012 Toyota Prius",
    "Charlotte",
    "red",
    "CarMax Financing"
  ],
  "attributes": [
    {
      "loanType": [
        "auto"
      ],
      "loanOfficers": [
        "Gabby Hayes",
        "Packer McGee"
      ],
      "securityLevel": [
        "proprietary"
      ]
    }
  ]
}
```

Response

The operation is judged to be successful (and returns a 200 status code) only if *all* of the requested owners were identified. If any of the owners couldn't be identified, the operation fails and returns 409.

HTTP Status Codes

If the call returns a 4xx or 5xx HTTP status, the response body will contain properties that provide more information about the error.

200	The Vault space was successfully reserved for all of the requested owners. The response body contains an assets array that contains an element for each owner that you specified in the request. Each element in the array contains the ownerId that identifies the owner, and a unique assetId that identifies the asset for that owner. Remember: Each owner of a given asset has its own asset identifier.
400	The request is missing a required property, one or more of the ownerId values in the request is invalid, or one or more of the customers that are identified through real-life attributes couldn't be found.
429	The request has been rejected because of rate limiting -- you've sent too many requests in a given amount of time.
500	General server error.

502 Internal connection failure.

503 The server is unavailable due to heavy traffic or maintenance.

Body Properties

assets **[object]**

The assets array contains an element for each owner that you specified in the request. Each element contains the owner's ownerId (which will have either been passed in in the request, or generated by this operation) and the assetId that identifies the owner's copy of the asset. Each element also contains an ownerIdInSourceSystem value. As explained in the owners property in the request, you use these values to correlate a real-world owner to the ownerId that the Vault has created for the owner.

ownerId **string**

An opaque string value that uniquely identifies an owner. If you want to add other assets to this owner, you should identify the owner through his or her ownerId rather than re-identifying the owner through personal attributes.

assetId **string**

An opaque string value that identifies the copy of the asset that was created for this owner. You use the assetId when you're pushing data (and metadata) into the Vault.

ownerIdInSourceSystem **string**

Brings back the value of the identically-named property that was passed in the request body.

JSON Structure

```
{
  "assets": [
    {
      "ownerId": "60abf667532ed9b90ba37",
      "assetId": "243a756109ad1034d2412",
      "ownerIdInSourceSystem": "dd7e2b018978476fe78c6"
    }
  ]
}
```

```
var https = require('https');
var queryString = require('querystring');

// Assemble the request message headers
var requestHeaders = {
  'Authorization': 'Bearer 7d113f4905c841a418401',
  'Content-Type': 'application/json',
  'Accept': 'application/json;v=0'
}

// Write the request body
requestBody = JSON.stringify({
  "owners": [
```

```
    {
      "ownerId": "a34e-ff87-6230-4875",
      "ownerIdInSourceSystem": "M101",
      "ownerDetails": {
        "phoneNumber": "8005551212",
        "email": "string",
        "individual": {
        },
        "address": {
        }
      }
    }
  ],
  "assetName": "Judy and Sam new car loan; bank statements, April-June
2017",
  "assetType": "application/pdf",
  "path": "/new_car/purchase/docs",
  "tags": [
    "2012 Toyota Prius",
    "Charlotte",
    "red",
    "CarMax Financing"
  ],
  "attributes": [
    {
      "loanType": [
        "auto"
      ],
      "loanOfficers": [
        "Gabby Hayes",
        "Packer McGee"
      ],
      "securityLevel": [
        "proprietary"
      ]
    }
  ]
})

// Assemble the calling options for the request message
var options = {
  method: 'POST',
  hostname: 'api-sandbox.capitalone.com',
  port: 443, // https
  path: '/vault/assets',
  headers: requestHeaders
}

// Create the request and handle the response
var reserveASpaceInTheVault = https.request(options, function(response)
{

  // Accumulate the response data
  var responseData = "";
  response.on('data', function(data) {
    responseData += data;
  });

  // Process the response data
  response.on('end', function() {
    // Do something with responseData
  });
});

// Write the request body into the request message
reserveASpaceInTheVault.write(requestBody);
```

```
// Finish sending the request
reserveASpaceInTheVault.end();
```

```
-- REQUEST --
curl -i -k --tlsv1 -H "Authorization: Bearer 7d113f4905c841a418401" -H
"Content-Type: application/json" -H "Accept: application/json;v=0" -d
'{"owners":[{"ownerId":"a34e-ff87-6230-
4875","ownerIdInSourceSystem":"M101","ownerDetails":
{"phoneNumber":"8005551212","email":"string","individual":{},"address":
{}}}],"assetName":"Judy and Sam new car loan; bank statements, April-
June
2017","assetType":"application/pdf","path":"/new_car/purchase/docs","tags
":["2012 Toyota Prius", "Charlotte", "red", "CarMax
Financing"],"attributes":[{"loanType": ["auto"], "loanOfficers": ["Gabby
Hayes", "Packer McGee"], "securityLevel": ["proprietary"] }]}' -X POST
"https://api-sandbox.capitalone.com/vault/assets"

--RESPONSE--
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 132
Date: Tue, 31 Oct 2017 12:54:09 GMT

{"assets":
[{"ownerId":"60abf667532ed9b90ba37","assetId":"243a756109ad1034d2412","ow
nerIdInSourceSystem":"dd7e2b018978476fe78c6"}]}
```


Push asset metadata into the Vault

The Push asset metadata into the Vault operation overwrites an asset's metadata with the values that you pass in the request body. If you want to update just a portion of the metadata — for example, if you want to add or remove a tag or attribute without affecting the rest of the metadata — you must retrieve the metadata through Retrieve asset metadata, modify the properties that are returned to you in that operation's response, and then use this operation to push the entire modified payload back to the Vault.

An asset's metadata comprises its name, type, path, attributes, and tags. It *doesn't* include the asset's owners. If you want to modify an asset's owners, you must reserve a new asset through Reserve a space in the Vault.

Keep in mind that the assetId that you specify in the URL identifies an asset that belongs to a specific owner. When you update that asset's metadata, you're only updating the metdata (on that asset) for that owner. If the asset has other owners, the metadata that's associated with those owners' assets isn't changed. If you want to keep the metadata objects in sync across all owners, you have to make separate (but equal) calls to this operation, passing in the assetId value for each owner in the successive calls.

```
PUT /vault/asset/{assetId}
```

Request

Path Parameters

assetId* **string**

The ID of the asset whose metadata you want to overwrite.

Headers

Authorization* **string**

The Bearer token that authenticates your app with Capital One. Instructions for constructing the token are given in Authorization.

Content-Type* **string**

Specifies the MIME type of the data in the request body. The setting should be exactly this:

```
Content-Type: application/json
```

Accept* **string**

Specifies the MIME type that you'll accept in the body of the response *and* declares the version of the endpoint that you're invoking. The setting should be exactly this:

```
Accept: application/json;v=0
```

Owner-ID* **string**

The ownerId of the customer on whose behalf this operation is being called. If the owner doesn't have sufficient permissions to upload the raw data, this operation will return 404.

Body Properties

Note	For descriptions of the request properties, see the identically-named properties in the Reserve a space in the Vault operation.

assetName* **string**

assetType **string** ***default: none***

path **string** ***default: none***

attributes **[object]**

tags **[string]** ***default: none***

JSON Structure

```
{
  "assetName": "Judy and Sam new car loan; bank statements, April-June
2017",
  "assetType": "application/pdf",
  "path": "/new_car/purchase/docs",
  "attributes": [
    {
      "loanType": [
        "auto"
      ],
      "loanOfficers": [
        "Gabby Hayes",
        "Packer McGee"
      ],
      "securityLevel": [
        "proprietary"
      ]
    }
  ],
  "tags": [
    "2012 Toyota Prius",
    "Charlotte",
    "red",
    "CarMax Financing"
  ]
}
```

Response

HTTP Status Codes

If the call returns a 4xx or 5xx HTTP status, the response body will contain properties that provide more information about the error.

204	The asset's metadata was successfully overwritten.
400	A property value in the request body is badly formed.
404	The client app or the owner identified by the Owner-ID request header doesn't have permission to call this operation, or the assetId path parameter is invalid. (For security reasons, the no-permission case and the invalid assetId case are purposefully indistinguishable.)
429	The request has been rejected because of rate limiting -- you've sent too many requests in a given amount of time.
500	Server error.
502	Internal connection failure.
503	The server is unavailable due to heavy traffic or maintenance.

```
var https = require('https');
var queryString = require('querystring');
```

```
// Assemble the request message headers
var requestHeaders = {
  'Authorization': 'Bearer d382da38b6adebb8a0d3b',
  'Content-Type': 'application/json',
  'Accept': 'application/json;v=0',
  'Owner-ID': 'a34e-ff87-6230-4875'
}

// URL encode the path parameter
assetId = encodeURIComponent(/* Unencoded assetId value */)

// Write the request body
requestBody = JSON.stringify({
  "assetName": "Judy and Sam new car loan; bank statements, April-June
2017",
  "assetType": "application/pdf",
  "path": "/new_car/purchase/docs",
  "attributes": [
    {
      "loanType": [
        "auto"
      ],
      "loanOfficers": [
        "Gabby Hayes",
        "Packer McGee"
      ],
      "securityLevel": [
        "proprietary"
      ]
    }
  ],
  "tags": [
    "2012 Toyota Prius",
    "Charlotte",
    "red",
    "CarMax Financing"
  ]
})

// Assemble the calling options for the request message
var options = {
  method: 'PUT',
  hostname: 'api-sandbox.capitalone.com',
  port: 443, // https
  path: '/vault/asset/' + assetId,
  headers: requestHeaders
}

// Create the request and handle the response
var pushAssetMetadataIntoTheVault = https.request(options,
function(response) {

  // Accumulate the response data
  var responseData = "";
  response.on('data', function(data) {
    responseData += data;
  });

  // Process the response data
  response.on('end', function() {
    // Do something with responseData
  });
});

// Write the request body into the request message
pushAssetMetadataIntoTheVault.write(requestBody);
```

```
// Finish sending the request
pushAssetMetadataIntoTheVault.end();
```

```
NOTE: {assetId} must be URL encoded.

-- REQUEST --
curl -i -k --tlsv1 -H "Authorization: Bearer d382da38b6adebb8a0d3b" -H
"Content-Type: application/json" -H "Accept: application/json;v=0" -H
"Owner-ID: a34e-ff87-6230-4875" -d '{"assetName":"Judy and Sam new car
loan; bank statements, April-June
2017","assetType":"application/pdf","path":"/new_car/purchase/docs","attr
ibutes":[{"loanType": ["auto"], "loanOfficers": ["Gabby Hayes", "Packer
McGee"], "securityLevel": ["proprietary"] }],"tags":["2012 Toyota
Prius", "Charlotte", "red", "CarMax Financing"]}' -X PUT "https://api-
sandbox.capitalone.com/vault/asset/{assetId}"

--RESPONSE--
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Length: 0
Date: Tue, 31 Oct 2017 12:54:09 GMT
```

Push asset data into the Vault

Pushes the raw data contained in the request body into the assets listed in the assetId query parameter.

Regardless of its actual data type, you must declare the MIME type of the request body to be `application/octet-stream` (see the Content-Type header). The data's actual MIME type is declared through the asset's assetType metadata property.

Important	This is an all-or-nothing operation. If you push data into multiple assets, each of the "pushes" must be successful, or else they all fail.

```
PUT /vault/loading-dock
```

Request

Query Parameters

assetId* **string**

A comma-separated list of assetId values that represent the assets (or "Vault spaces") into which you want to push the data that's contained in the request body. assetId values are created and returned to

you when you call the Reserve a space in the Vault operation.

Headers

Authorization* **string**

The Bearer token that authenticates your app with Capital One. Instructions for constructing the token are given in Authorization.

Owner-ID* **string**

The ownerId of the customer on whose behalf this operation is being called. If the owner doesn't have sufficient permissions to upload the raw data, this operation will return 404.

Content-Type* **string**

Specifies the MIME type of the data in the request body. The setting should be exactly this:

```
Content-Type: application/octet-stream
```

Payload

application/octet-stream

The asset's raw data.

Response

HTTP Status Codes

If the call returns a 4xx or 5xx HTTP status, the response body will contain properties that provide more information about the error.

204 The data was successfully pushed into the Vault.

404 The client app or the owner identified by the Owner-ID request header doesn't have permission to call this operation, or one or more of the values in the assetId query parameter is invalid. (For security reasons, the no-permission case and the invalid assetId case are purposefully indistinguishable.)

Important	If you're pushing data into multiple asset and one or more of the assetId values that identify those assets is invalid, the data won't be pushed into *any* of the assets.

429 The request has been rejected because of rate limiting -- you've sent too many requests in a given amount of time.

500	Server error.
502	Internal connection failure.
503	The server is unavailable due to heavy traffic or maintenance.

```
var https = require('https');
var queryString = require('querystring');

// Assemble the request message headers
var requestHeaders = {
  'Authorization': 'Bearer d92485001e1cbe9489fe2',
  'Owner-ID': 'a34e-ff87-6230-4875',
  'Content-Type': 'application/octet-stream'
}

// Assemble the query parameters
var queryParams = {
  'assetId': 'string'
}

// Write the request body
requestBody = "2550 4446 2d31 2e34 0a31 2030 206f 626a..."

// Assemble the calling options for the request message
var options = {
  method: 'PUT',
  hostname: 'api-sandbox.capitalone.com',
  port: 443, // https
  path: '/vault/loading-dock' + queryString.stringify(queryParams),
  headers: requestHeaders
}

// Create the request and handle the response
var pushAssetDataIntoTheVault = https.request(options,
function(response) {

  // Accumulate the response data
  var responseData = "";
  response.on('data', function(data) {
    responseData += data;
  });

  // Process the response data
  response.on('end', function() {
    // Do something with responseData
  });
});

// Write the request body into the request message
pushAssetDataIntoTheVault.write(requestBody);

// Finish sending the request
pushAssetDataIntoTheVault.end();
```

```
-- REQUEST --
curl -i -k --tlsv1 -H "Authorization: Bearer d92485001e1cbe9489fe2" -H
"Owner-ID: a34e-ff87-6230-4875" -H "Content-Type: application/octet-
```

```
stream" -d 2550 4446 2d31 2e34 0a31 2030 206f 626a... -X PUT
"https://api-sandbox.capitalone.com/vault/loading-dock?assetId=string"

--RESPONSE--
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Length: 0
Date: Tue, 31 Oct 2017 12:54:09 GMT
```

Retrieve asset metadata from the Vault

Important	This operation isn't currently supported.

Returns the metadata that's part of the asset that's identified by the assetId path parameter. The response also returns a status property that you can refer to to determine whether or not the asset's data can be downloaded.

`GET /vault/asset/{assetId}`

Request

Path Parameters

assetId* **string**

The ID of the asset whose metadata you want to overwrite.

Headers

Authorization* **string**

The Bearer token that authenticates your app with Capital One. Instructions for constructing the token are given in Authorization.

Accept* **string**

Specifies the MIME type that you'll accept in the body of the response *and* declares the version of the endpoint that you're invoking. The setting should be exactly this:

```
Accept: application/json;v=0
```

Owner-ID* **string**

The ownerId of the customer on whose behalf this operation is being called. If the owner doesn't have sufficient permissions to upload the raw data, this operation will return 404.

Response

HTTP Status Codes

If the call returns a 4xx or 5xx HTTP status, the response body will contain properties that provide more information about the error.

200 The asset's metadata was successfully retrieved.

404 The client app or the owner identified by the Owner-ID request header doesn't have permission to call this operation, or the assetId path parameter is invalid. (For security reasons, the no-permission case and the invalid assetId case are purposefully indistinguishable.)

429 The request has been rejected because of rate limiting -- you've sent too many requests in a given amount of time.

500 Server error.

502 Internal connection failure.

503 The server is unavailable due to heavy traffic or maintenance.

Body Properties

ownerId **string**

A token that uniquely identifies this asset's owner. Keep in mind that if more than one person is declared as an asset's owner, each of these owners addresses the asset through his or her own assetId. Thus, the ownerId that's brought back, here, corresponds to the assetId that was passed in the URL.

assetId **string**

The asset's assetId. This will be the same as the assetId value that you provided as a path parameter in the URL.

status **[string]**

When an asset's data is uploaded into the Vault, it has to go through some processing before it's available for download. The status property proclaims the stage of this processing:

- `None`. The asset doesn't contain any data.
- `InProgress`. The data hasn't been fully processed. You shouldn't depend on the accuracy or completeness of the data.
- `Complete`. The data has been fully processed and can be safely downloaded.

Note	For descriptions of the rest of the response properties, see the identically-named properties in the Reserve a space in the Vault operation.

assetName* **string**

assetType **string**

path **string**

attributes **[object]**

tags **[string]**

JSON Structure

```
{
  "ownerId": "3131a01b168131bb34124",
  "assetId": "01d59c3dc187999609361",
  "status": [
    "Complete"
  ],
  "assetName": "Judy and Sam new car loan; bank statements, April-June
2017",
  "assetType": "application/pdf",
  "path": "/new_car/purchase/docs",
  "attributes": [
    {
      "loanType": [
        "auto"
      ],
      "loanOfficers": [
        "Gabby Hayes",
        "Packer McGee"
      ],
      "securityLevel": [
        "proprietary"
      ]
    }
  ],
  "tags": [
    "2012 Toyota Prius",
    "Charlotte",
    "red",
    "CarMax Financing"
  ]
}
```



```
var https = require('https');
var queryString = require('querystring');

// Assemble the request message headers
var requestHeaders = {
  'Authorization': 'Bearer c3e9e87574413b761defe',
  'Accept': 'application/json;v=0',
  'Owner-ID': 'a34e-ff87-6230-4875'
}
```

```
// URL encode the path parameter
assetId = encodeURIComponent(/* Unencoded assetId value */)

// Assemble the calling options for the request message
var options = {
  method: 'GET',
  hostname: 'api-sandbox.capitalone.com',
  port: 443, // https
  path: '/vault/asset/' + assetId,
  headers: requestHeaders
}

// Create the request and handle the response
var retrieveAssetMetadataFromTheVault = https.request(options,
function(response) {

  // Accumulate the response data
  var responseData = "";
  response.on('data', function(data) {
    responseData += data;
  });

  // Process the response data
  response.on('end', function() {
    // Do something with responseData
  });
});

// Finish sending the request
retrieveAssetMetadataFromTheVault.end();
```

```
NOTE: {assetId} must be URL encoded.

-- REQUEST --
curl -i -k --tlsv1 -H "Authorization: Bearer c3e9e87574413b761defe" -H
"Accept: application/json;v=0" -H "Owner-ID: a34e-ff87-6230-4875" -X GET
"https://api-sandbox.capitalone.com/vault/asset/{assetId}"

--RESPONSE--
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 425
Date: Tue, 31 Oct 2017 12:54:09 GMT

{"ownerId":"3131a01b168131bb34124","assetId":"01d59c3dc187999609361","sta
tus":["Complete"],"assetName":"Judy and Sam new car loan; bank
statements, April-June
2017","assetType":"application/pdf","path":"/new_car/purchase/docs","attr
ibutes":[{"loanType": ["auto"], "loanOfficers": ["Gabby Hayes", "Packer
McGee"], "securityLevel": ["proprietary"] }],"tags":["2012 Toyota
Prius", "Charlotte", "red", "CarMax Financing"]}
```


Retrieve asset data from the Vault

Important	This operation isn't currently supported.

Retrieves the asset's data as raw, octet-stream bytes. The response message doesn't declare the MIME type of the data. It's expected that you will have declared the MIME type in the assetType metadata property, or that you otherwise know how to interpret the data based on some other metadata tag or attributes.

To improve performance and lower bandwidth, this operation lets you compress the response and (potentially) skip the response altogether (because you already have it):

- Accept-Encoding lets you ask that the response payload be compressed. Currently, the only compression algorithm you can request is `gzip, deflate`.
- When you request an asset's data, the response includes an ETag header. The next time you ask for the same data, you can pass the ETag value in the If-None-Match request header. If the data hasn't changed since the ETag was generated, the operation will return a _304 HTTP status and an empty response body. You would then use the payload from the previous call (which we'll assume you have cached).

```
GET /vault/assets/{assetId}/raw
```

Request

Path Parameters

assetId* **string**

The ID of the asset whose data you want to retrieve.

Headers

Authorization* **string**

The Bearer token that authenticates your app with Capital One.

Accept* **string**

Specifies the MIME type that you'll accept in the body of the response *and* declares the version of the endpoint that you're invoking. The setting should be exactly this:

```
Accept: application/octet-stream;v=1
```

Accept-Encoding **string** ***default: none***

Declares the algorithm that will be used to compress the data in the response payload. For most binary file types (videos, jpeg), compression won't have any positive effect but it may help with large non-binary files such as spreadsheets. or some PPT, PDF files etc.).

Currently, the only value you should use (if you use any at all) is:

```
Accept-Encoding: gzip, deflate
```

If-None-Match **string** ***default: none***

A list of `ETags` (although you typically only pass one) that can be used to determine if the response needs to be sent back. See the overview for this operation for more information.

Response

HTTP Status Codes

If the call returns a 4xx or 5xx HTTP status, the response body will contain properties that provide more information about the error.

200 The asset's data was successfully returned in the response body.

304 The data hasn't been modified since the last time the client requested it (as determined by the ETag value that was passed in the If-None-Match request header); the response body is empty. See the overview for this operation for more information.

404 The client app or the owner identified by the ownerId query parameter doesn't have permission to call this operation, or one or more of the values in the assetId query parameter is invalid. (For security reasons, the no-permission case and the invalid assetId case are purposefully indistinguishable.)

429 The request has been rejected because of rate limiting -- you've sent too many requests in a given amount of time.

500 Server error.

502 Internal connection failure.

503 The server is unavailable due to heavy traffic or maintenance.

ETag **string** A value that represents the current state of the asset's data. See the overview of the operation for more information.

Payload

application/octet-stream

The asset's data.

```
var https = require('https');
var queryString = require('querystring');

// Assemble the request message headers
var requestHeaders = {
  'Authorization': 'string',
```

```
  'Accept': 'application/octet-stream;v=1',
  'Accept-Encoding': 'gzip, deflate',
  'If-None-Match': '6e84ff2ca85f083a8541c'
}

// URL encode the path parameter
assetId = encodeURIComponent(/* Unencoded assetId value */)

// Assemble the calling options for the request message
var options = {
  method: 'GET',
  hostname: 'api-sandbox.capitalone.com',
  port: 443, // https
  path: '/vault/assets/' + assetId + '/raw',
  headers: requestHeaders
}

// Create the request and handle the response
var retrieveAssetDataFromTheVault = https.request(options,
function(response) {

  // Accumulate the response data
  var responseData = "";
  response.on('data', function(data) {
    responseData += data;
  });

  // Process the response data
  response.on('end', function() {
    // Do something with responseData
  });
});

// Finish sending the request
retrieveAssetDataFromTheVault.end();
```

```
NOTE: {assetId} must be URL encoded.

-- REQUEST --
curl -i -k --tlsv1 -H "Authorization: string" -H "Accept:
application/octet-stream;v=1" -H "Accept-Encoding: gzip, deflate" -H
"If-None-Match: 6e84ff2ca85f083a8541c" -X GET "https://api-
sandbox.capitalone.com/vault/assets/{assetId}/raw"

--RESPONSE--
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 42
Date: Tue, 31 Oct 2017 12:54:09 GMT

2550 4446 2d31 2e34 0a31 2030 206f 626a...
```


Search for assets in the Vault

Important	This operation isn't currently supported.

The Search for assets in the Vault operation lets you search for assets based on the values that are recorded in the assets' metadata. Specifically, you can search for an asset by its…

- owner
- name
- data type
- path
- tags
- attributes

You *can't* search within the content of an asset's data, but you can ask for assets whose data was created or modified within a given date range.

The operation returns an array of the assetId values that fulfill the search criteria. If no assets are found, an empty array is returned (but the operation is still considered to have succeeded).

The search is performed by AND'ing all of search properties that you specify in the request body. For example, if you specify a tags criterion and a path criterion, the operation will only return assets within the given path that contain the specified tags.

Search expressions

The way you express a search criterion depends on the metadata property that you're searching. The properties and their search expressions are:

- owner takes an array of ownerId values; an asset must be owned by one (but not all) of the owners that you specify.
- assetName takes a "nested boolean hash" object. The hash contains one or both of the `#any` and `#all` keys; the corresponding value is an array where each element is either **a)** a word or phrase that's matched as a case-insensitive substring of the asset's name, or **b)** a nested boolean hash. For example:

```
"assetName" :
{
  "#all" : ["Jim", "Elly Jane"],
  "#any" : ["car", "home",
    {
      "#all" : ["Washington", "Tuesday"]
    }
  ]
}
```

  This expression means "Find assets whose names contain both 'jim' and 'elly jane' and that also include either 'car', 'home', or both 'washington' and 'tuesday'.

- assetType takes an array of candidate MIME types.

- path takes a single pathname value that's consider the root of the search. For example, if you specify a path value of `/archive`, the assets that are within the `/archive` directory and its subdirectories are considered.

- created and updated take two-element tuples (arrays) where the first element is a start date and the second element is an end date. The dates are expressed as ISO 8601 values. You can use "*"" to mean any time in the past and any time up to the present, respectively for the two elements (an example is given in the next section).

  Keep in mind that the dates ranges apply to the time that the asset's *data* — not its metadata or the "asset space" itself — was created and modified.

- tags takes a nested boolean hash object, similar to assetName. As with assetName, the array elements in the hash value are case-insensitive; unlike assetName, the values are not compared as substrings. If you search for an asset that has a tag value of `auto`, you won't get assets that are tagged with `automobile`.

- attributes also takes a nested boolean hash object where the hash value array elements are "key: value"" pairs that give the key and value of the desired attribute.

Example

Consider the following request body example:

```
{
  "owner": ["cd4869a9", "8be65be4"]
  "path": "/archive",
  "updated": ["2001-04-12T23:20:50.52Z", "*"],
  "created": ["*", "1997-01-01T01:00:00.00Z"],
  "attributes": { "#any" : [
    {"#all" : [{"type": "legal"}, {"source": "A, B, C & Associates"}]},
    {"#all" : [{"type": "car loan"}, {"loan officer": "Joanne Smith"}]}
  ]},
  "tags": { "#any" : [
    {"#all" : ["yellow"]},
    {"#all" : ["green", "blue"]}
  ]},

}
```

This will find assets that fulfill *all* of the following:

- The asset is owned by "cd4869a9" OR "8be65be4"
- The data is stored within the `/archive` directory or any of its subdirectories
- The data was created before Jan 1st, 1997 and was updated after April 12th, 2001
- Its `type` attribute is set to `legal` and its `source` attribute is set to `A, B, C & Associates` OR `type` == `car loan` and `loan officer` == `Joanne Smith`
- It's tagged with `yellow` OR both `green` and `blue`

```
POST /vault/query
```

Request

Headers

Authorization* **string**

The Bearer token that authenticates your app with Capital One. Instructions for constructing the token are given in Authorization.

Content-Type* **string**

Specifies the MIME type of the data in the request body. The setting should be exactly this:

```
Content-Type: application/json
```

Accept* **string**

Specifies the MIME type that you'll accept in the body of the response *and* declares the version of the endpoint that you're invoking. The setting should be exactly this:

```
Accept: application/json;v=0
```

Body Properties

See the overview to this operation for information about the body properties.

owner **[string]** ***default: none***

An array of ownerId values. An asset must be owned by at least one of the listed owners.

assetName **object** ***default: none***

A "nested boolean hash" that contain words or phrases that must appear in the asset's name.

assetType **[string]** ***default: none***

An array of MIME types, one of which the asset's data must conform to (as designated by the asset's assetType metadata).

path **string**

The pathname of the directory in which the asset's data must be stored. This includes subdirectories.

attributes **[[object]]** ***default: none***

list of attributes joined by AND

tags **[[string]]** ***default: none***

list of tags joined by AND

created **[string, string]** ***default: none***

A two-element array of ISO 8601 values that defines the time span during which the asset's data must have been pushed to the Vault.

updated **[string, string]** ***default: none***

A two-element array of ISO 8601 values that defines the time span during which the asset's data must have been updated.

JSON Structure

```
{
  "owner": [
    "string"
  ],
  "assetName": {
  },
  "assetType": [
    "string"
  ],
  "path": "string",
  "attributes": [

  ],
  "tags": [

  ],
  "created": [

  ],
  "updated": [

  ]
}
```

Response

HTTP Status Codes

If the call returns a 4xx or 5xx HTTP status, the response body will contain properties that provide more information about the error.

200 The operation completed successfully. The assets that were found are returned in the assets array. If no assets fulfilled the search criteria, the array will be empty (but present).

400 A property value in the request is badly formed.

429 The request has been rejected because of rate limiting -- you've sent too many requests in a given amount of time.

500 Server error.

502 Internal connection failure.

503 The server is unavailable due to heavy traffic or maintenance.

Body Properties

assets **[object]**

An array of objects that describe the assets that passed the search criteria.

> **Note** For descriptions of the response properties, see the Retrieve asset metadata from the Vault operation.

ownerId **string**

assetId **string**

status **[string]**

assetName* **string**

assetType **string**

path **string**

attributes **[object]**

tags **[string]**

JSON Structure

```
{
  "assets": [
    {
      "ownerId": "IDPLEASE",
      "assetId": "2416f0608fcfbb1699b05",
      "status": [
        "Complete"
      ],
      "assetName": "Judy and Sam new car loan; bank statements, April-
June 2017",
      "assetType": "application/pdf",
      "path": "/new_car/purchase/docs",
      "attributes": [
        {
          "loanType": [
            "auto"
          ],
          "loanOfficers": [
            "Gabby Hayes",
            "Packer McGee"
          ],
          "securityLevel": [
            "proprietary"
          ]
```

```
        }
      ],
      "tags": [
        "2012 Toyota Prius",
        "Charlotte",
        "red",
        "CarMax Financing"
      ]
    }
  ]
}
```

```
var https = require('https');
var queryString = require('querystring');

// Assemble the request message headers
var requestHeaders = {
  'Authorization': 'Bearer 25eca833abf0a5ac67741',
  'Content-Type': 'application/json',
  'Accept': 'application/json;v=0'
}

// Write the request body
requestBody = JSON.stringify({
  "owner": [
    "string"
  ],
  "assetName": {
  },
  "assetType": [
    "string"
  ],
  "path": "string",
  "attributes": [

  ],
  "tags": [

  ],
  "created": [

  ],
  "updated": [

  ]
})

// Assemble the calling options for the request message
var options = {
  method: 'POST',
  hostname: 'api-sandbox.capitalone.com',
  port: 443, // https
  path: '/vault/query',
  headers: requestHeaders
}

// Create the request and handle the response
var searchForAssetsInTheVault = https.request(options,
function(response) {

  // Accumulate the response data
  var responseData = "";
  response.on('data', function(data) {
    responseData += data;
```

```
  });

  // Process the response data
  response.on('end', function() {
    // Do something with responseData
  });
});

// Write the request body into the request message
searchForAssetsInTheVault.write(requestBody);

// Finish sending the request
searchForAssetsInTheVault.end();
```

```
-- REQUEST --
curl -i -k --tlsv1 -H "Authorization: Bearer 25eca833abf0a5ac67741" -H
"Content-Type: application/json" -H "Accept: application/json;v=0" -d
'{"owner":["string"],"assetName":{},"assetType":
["string"],"path":"string","attributes":[],"tags":[],"created":
[],"updated":[]}' -X POST "https://api-
sandbox.capitalone.com/vault/query"

--RESPONSE--
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 425
Date: Tue, 31 Oct 2017 12:54:09 GMT

{"assets":
[{"ownerId":"IDPLEASE","assetId":"2416f0608fcfbb1699b05","status":
["Complete"],"assetName":"Judy and Sam new car loan; bank statements,
April-June
2017","assetType":"application/pdf","path":"/new_car/purchase/docs","attr
ibutes":[{"loanType": ["auto"], "loanOfficers": ["Gabby Hayes", "Packer
McGee"], "securityLevel": ["proprietary"] }],"tags":["2012 Toyota
Prius", "Charlotte", "red", "CarMax Financing"]}]}
```


Grant permissions on assets

Important	This operation isn't currently supported.

Sometimes you want to share an asset with another individual, but you don't want them to be able to share it or update it. Other times you want to allow another owner to update, delete, categorize, the whole spectrum. Grant permissions on assets allows you to add permissions for owners beyond the original asset owner.

You need a couple of items to make this operation work:

- An ownerId for an owner with `grant` permission
- The individual ownerIds for the owners you are granting permissions to

- The assetIds for the assets to grant permissions for

You can grant permission for more than one asset in one call, however all permissions included in the request are applied to all assets in the request.

Permission types include:

- `grant` - allows owner to grant permissions to other users for that specific asset
- `read` - allows owner to view the asset
- `categorize` - allows owner to manage metadata and tags
- `update` - allows owner to modify the asset's binary data
- `delete` - allows owner to delete the asset

No partial success is allowed. If any one permission fails, none will succeed. The response includes an error code to help you pinpoint the error cause.

`PUT /vault/authorization/grant`

Request

Headers

Authorization* **string**

The Bearer token that authenticates your app with Capital One. Instructions for constructing the token are given in Authorization.

Content-Type* **string**

Specifies the MIME type of the data in the request body. The setting should be exactly this:

```
Content-Type: application/json
```

Accept* **string**

Specifies the MIME type that you'll accept in the body of the response *and* declares the version of the endpoint that you're invoking. The setting should be exactly this:

```
Accept: application/json;v=0
```

Owner-ID* **string**

The ownerId of the customer on whose behalf this operation is being called. If the owner doesn't have sufficient permissions to upload the raw data, this operation will return 404.

Body Properties

permissions* **[string]**

An array of permission types. These permissions will be granted on all assets include in the request. Permission types include:

- `grant` - allows user to grant permissions to other users for that specific asset
- `read` - allows user to view the asset
- `categorize` - allows user to manage metadata and tags
- `update` - allows user to modify the asset's binary data
- `delete` - allows user to delete the asset

securedObjects* **[string]**

An array of assetIds for the assets you want to grant specified permissions on.

subjects* **[string]**

The owners to grant permissions for, included as an array of ownerIds.

Note	All permissions for an original owner of the asset are included at asset creation.

JSON Structure

```
{
  "permissions": [
    "read",
    "write"
  ],
  "securedObjects": [
    "12345 , 23456"
  ],
  "subjects": [
    "12768723683746, 12768723683747"
  ]
}
```

Response

HTTP Status Codes

If the call returns a 4xx or 5xx HTTP status, the response body will contain properties that provide more information about the error.

204	The permssions were granted successfully. The response body is empty.
400	One or more of the values in the securedObjects and subjects arrays in the request message is

	invalid.
404	The client app or the owner identified by the Owner-ID request header doesn't have permission to call this operation.
429	The request has been rejected because of rate limiting -- you've sent too many requests in a given amount of time.
500	Server error.
502	Internal connection failure.
503	The server is unavailable due to heavy traffic or maintenance.

```
var https = require('https');
var queryString = require('querystring');

// Assemble the request message headers
var requestHeaders = {
  'Authorization': 'Bearer 197ba1034fffb62ac3943',
  'Content-Type': 'application/json',
  'Accept': 'application/json;v=0',
  'Owner-ID': 'a34e-ff87-6230-4875'
}

// Write the request body
requestBody = JSON.stringify({
  "permissions": [
    "read",
    "write"
  ],
  "securedObjects": [
    "12345 , 23456"
  ],
  "subjects": [
    "12768723683746, 12768723683747"
  ]
})

// Assemble the calling options for the request message
var options = {
  method: 'PUT',
  hostname: 'api-sandbox.capitalone.com',
  port: 443, // https
  path: '/vault/authorization/grant',
  headers: requestHeaders
}

// Create the request and handle the response
var grantPermissionsOnAssets = https.request(options, function(response)
{

  // Accumulate the response data
  var responseData = "";
  response.on('data', function(data) {
    responseData += data;
  });

  // Process the response data
  response.on('end', function() {
    // Do something with responseData
  });
```

```
});

// Write the request body into the request message
grantPermissionsOnAssets.write(requestBody);

// Finish sending the request
grantPermissionsOnAssets.end();
```

```
-- REQUEST --
curl -i -k --tlsv1 -H "Authorization: Bearer 197ba1034fffb62ac3943" -H
"Content-Type: application/json" -H "Accept: application/json;v=0" -H
"Owner-ID: a34e-ff87-6230-4875" -d '{"permissions":["read",
"write"],"securedObjects":["12345 , 23456"],"subjects":["12768723683746,
12768723683747"]}' -X PUT "https://api-
sandbox.capitalone.com/vault/authorization/grant"

--RESPONSE--
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Length: 0
Date: Tue, 31 Oct 2017 12:54:09 GMT
```

Revoke permissions on assets

Important	This operation isn't currently supported.

Use this operation to revoke permissions previously granted to users other than the original asset owner. Similar to Grant permissions on assets, any permission types you pass in the request are revoked for *all* assetIds you specify. If you need to revoke different permissions for each assetId, you should call this operation individually for each assetId.

```
PUT /vault/authorization/revocation
```

Request

Headers

Authorization* **string**

The Bearer token that authenticates your app with Capital One. Instructions for constructing the token are given in Authorization.

Content-Type* **string**

Specifies the MIME type of the data in the request body. The setting should be exactly this:

```
Content-Type: application/json
```

Accept* **string**

Specifies the MIME type that you'll accept in the body of the response *and* declares the version of the endpoint that you're invoking. The setting should be exactly this:

```
Accept: application/json;v=0
```

Owner-ID* **string**

The ownerId of the customer on whose behalf this operation is being called. If the owner doesn't have sufficient permissions to upload the raw data, this operation will return 404.

Body Properties

permissions* **[string]**

An array of permission types to revoke. These permissions will be revoked on all assets included in the request. You do not have to revoke all permissions currently in place on an asset. For example, you can revoke a user's `grant` permission, but not their `read` access on an asset by only including `grant` in this array.

Permission types include:

- `grant` - allows user to grant permissions to other users for that specific asset
- `read` - allows user to view the asset
- `categorize` - allows user to manage metadata and tags
- `update` - allows user to modify the asset's binary data
- `delete` - allows user to delete the asset

securedObjects* **[string]**

An array of assetIds for the assets you want to revoke specified permissions on.

subjects* **[string]**

The owners to revoke permissions for, included as an array of ownerIds.

Note	Permissions for an original owner of the asset cannot be revoked.

JSON Structure

```
{
  "permissions": [
    "read",
    "write"
  ],
  "securedObjects": [
    "12345 , 23456"
  ],
  "subjects": [
    "12768723683746, 12768723683747"
  ]
}
```

Response

HTTP Status Codes

If the call returns a 4xx or 5xx HTTP status, the response body will contain properties that provide more information about the error.

204	The specified permissions were revoked successfully. The response body is empty.
400	One or more of the values in the securedObjects and subjects arrays in the request message is invalid.
404	The client app or the owner identified by the Owner-ID request header doesn't have permission to call this operation.
429	The request has been rejected because of rate limiting -- you've sent too many requests in a given amount of time.
500	Server error.
502	Internal connection failure.
503	The server is unavailable due to heavy traffic or maintenance.


```
var https = require('https');
var queryString = require('querystring');

// Assemble the request message headers
var requestHeaders = {
   'Authorization': 'Bearer abb60f27d7368b5c9caa5',
   'Content-Type': 'application/json',
   'Accept': 'application/json;v=0',
   'Owner-ID': 'a34e-ff87-6230-4875'
}

// Write the request body
requestBody = JSON.stringify({
   "permissions": [
     "read",
     "write"
   ],
```

```
  "securedObjects": [
    "12345 , 23456"
  ],
  "subjects": [
    "12768723683746, 12768723683747"
  ]
})

// Assemble the calling options for the request message
var options = {
  method: 'PUT',
  hostname: 'api-sandbox.capitalone.com',
  port: 443, // https
  path: '/vault/authorization/revocation',
  headers: requestHeaders
}

// Create the request and handle the response
var revokePermissionsOnAssets = https.request(options,
function(response) {

  // Accumulate the response data
  var responseData = "";
  response.on('data', function(data) {
    responseData += data;
  });

  // Process the response data
  response.on('end', function() {
    // Do something with responseData
  });
});

// Write the request body into the request message
revokePermissionsOnAssets.write(requestBody);

// Finish sending the request
revokePermissionsOnAssets.end();
```

```
-- REQUEST --
curl -i -k --tlsv1 -H "Authorization: Bearer abb60f27d7368b5c9caa5" -H
"Content-Type: application/json" -H "Accept: application/json;v=0" -H
"Owner-ID: a34e-ff87-6230-4875" -d '{"permissions":["read",
"write"],"securedObjects":["12345 , 23456"],"subjects":["12768723683746,
12768723683747"]}' -X PUT "https://api-
sandbox.capitalone.com/vault/authorization/revocation"

--RESPONSE--
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Length: 0
Date: Tue, 31 Oct 2017 12:54:09 GMT
```