ALTS
Alabama Titling System

ALTS API Overview

ALTS Webservice
The ALTS service is a REST-based service used to interact with ALTS without visiting the ALTS website.

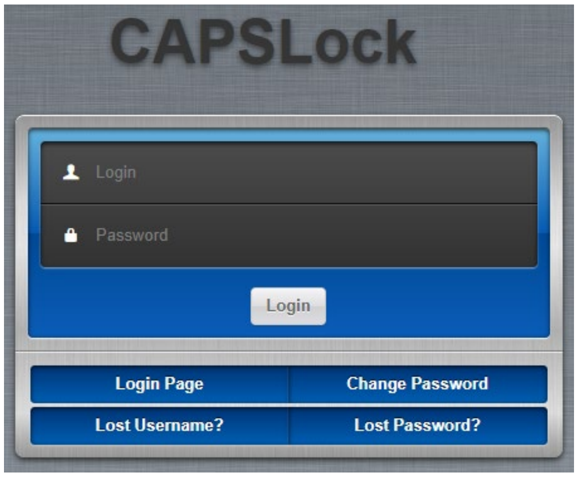
Development Location:
API Access Portal/Token Acquisition: https://apiregistration-uat.mvtrip.alabama.gov/
API: https://altsapi-uat.mvtrip.alabama.gov/api/v1.0/
Token API Endpoint: https://capslockoidc-uat.mvtrip.alabama.gov/connect/token

Production Location:
API Access Portal/Token Acquisition: https://apiregistration.mvtrip.alabama.gov/
API: https://altsapi.mvtrip.alabama.gov/api/v1.0/
Token API Endpoint: https://capslockoidc.mvtrip.alabama.gov/connect/token

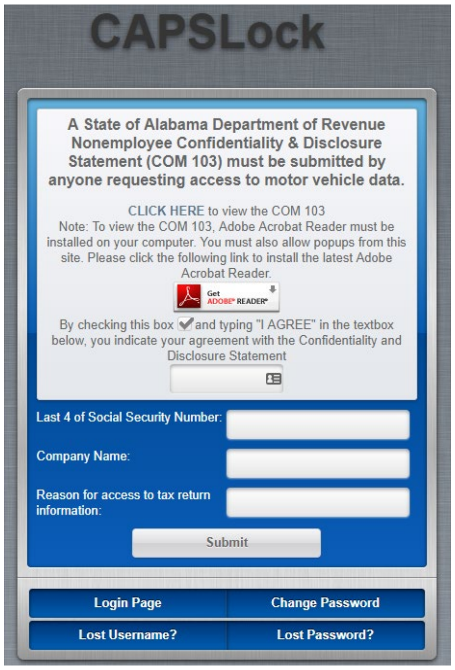
Service Authentication
The service authentication mechanism used by our APIs is OpenID Connect (https://openid.net/specs/openid-connect-core-1_0-final.html). To securely connect to any of our APIs, your application will need to pass an access token via a standard HTTP request header; otherwise, your application will receive an HTTP 401 unauthorized response. Your application will also need to obtain a refresh token for cases where the access token expires. The refresh token can be used to obtain a new access token without requiring user interaction. Storage for both of these tokens should be as secure as possible since access to them means direct access to our APIs. Below are the steps required to obtain, maintain, and use access and refresh tokens.

Obtaining Tokens
Steps to obtain tokens:

1. Have the user of the application go to the MVTRIP API access portal and login with their MVTRIP credentials. The URLs are above, and the correct one to use depends on which environment you need credentials and/or tokens for.



Once you log in for the first time, you may be asked to agree to the ADOR Confidentiality and Disclosure

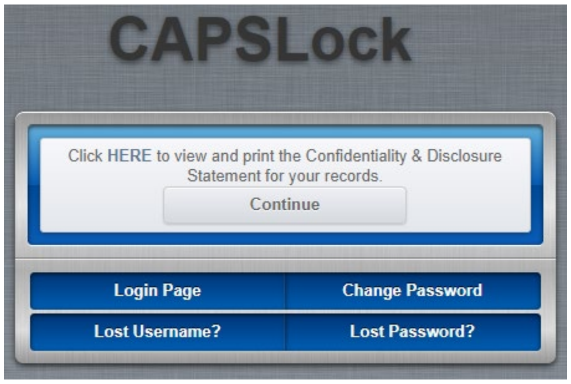


A State of Alabama Department of Revenue Nonemployee Confidentiality & Disclosure Statement (COM 103) must be submitted by anyone requesting access to motor vehicle data.

CLICK HERE to view the COM 103
Note: To view the COM 103, Adobe Acrobat Reader must be installed on your computer. You must also allow popups from this site. Please click the following link to install the latest Adobe Acrobat Reader.

By checking this box ✓ and typing "I AGREE" in the textbox below, you indicate your agreement with the Confidentiality and Disclosure Statement

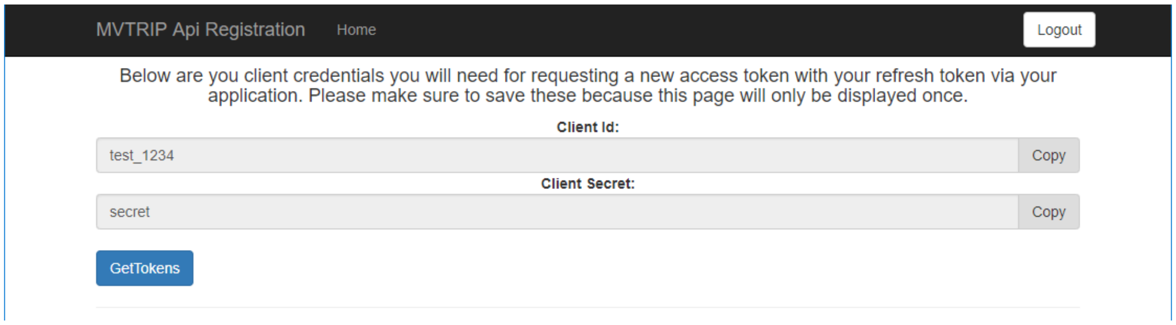
Last 4 of Social Security Number:
Company Name:
Reason for access to tax return information:



Click HERE to view and print the Confidentiality & Disclosure Statement for your records.

2. Once authenticated, they will need to generate a new client id and password for their client by clicking the Generate client credentials button. If they have already done this and are simply requesting new tokens then you can skip to step 4.

3. After the client id and password have been generated have the user click the Copy button for each value and paste it into the appropriate form in your application.

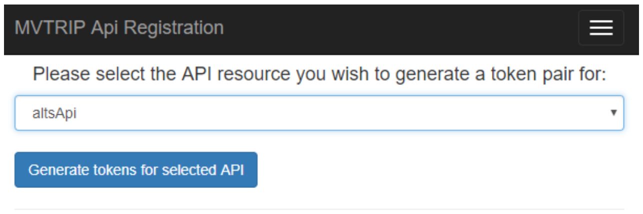


Below are you client credentials you will need for requesting a new access token with your refresh token via your application. Please make sure to save these because this page will only be displayed once.

Client Id:
test_1234

Client Secret:
secret

4. After obtaining and setting the client credentials have the user click the GetTokens button. You will be asked which API you wish to generate tokens for. Select "altsApi"



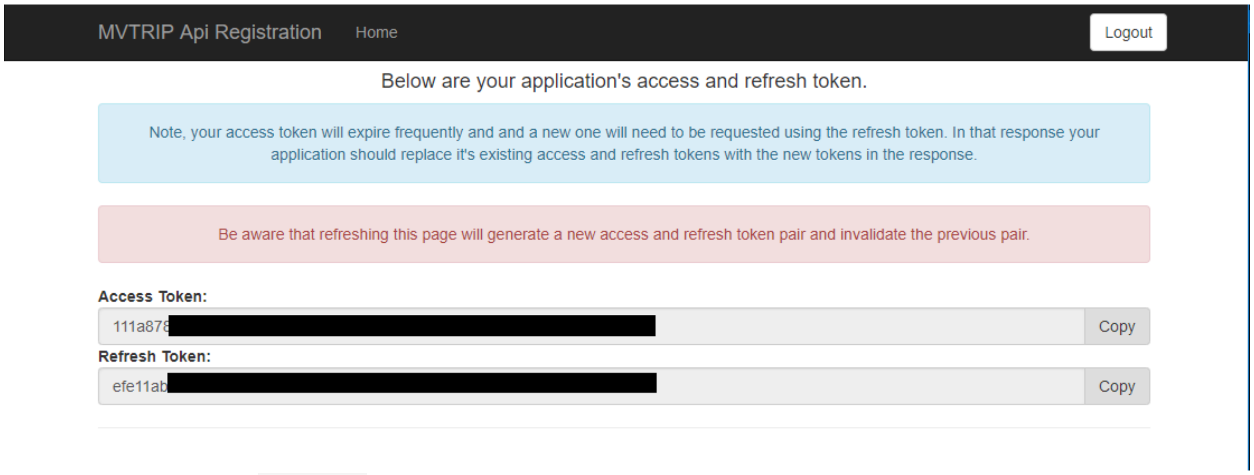
Please select the API resource you wish to generate a token pair for:

altsApi

and click the Generate tokens for selected API button.

Note, if they already have a valid set of access and refresh tokens, the system will warn them that proceeding with the request will invalidate the old tokens.

5. Once the tokens have been generated, you should see something like this



Below are your application's access and refresh token.

Note, your access token will expire frequently and and a new one will need to be requested using the refresh token. In that response your application should replace it's existing access and refresh tokens with the new tokens in the response.

Be aware that refreshing this page will generate a new access and refresh token pair and invalidate the previous pair.

Access Token:

111a87[redacted]

Refresh Token:

efe11ab[redacted]

The user should click Copy for each token and then paste it into the appropriate form in your application. These tokens will only be displayed once. After the user navigates away from this page they will no longer be able to see their tokens.

A few things to note about obtaining tokens:

- Your application will need some form the user can paste the access and refresh tokens into, as well as the client credentials. This is probably best done on install and in the settings/configuration since these values can change, though they should not very frequently (see the next bullet).
- Access tokens are valid for 8 hours after issuance, and refresh tokens are valid for 30 days. Once an access token expires, any request using it will receive a 401 unauthorized response. At this point, your application should use the refresh token obtained above to get a new access and refresh token pair. As long as the token is refreshed every 30 days, the user should be able to avoid going back to the website to obtain a new pair.
- Tokens should be obtained by the user who is performing the actions in your application, and NOT be shared across one user account. Every method called on our APIs will be logged as the user who obtained the token.
- Our APIs will determine access rights on the requested method by checking the privilege level of the user who obtained the token.

Access token usage

To call any of our APIs, you will need to pass a valid access token in the HTTP request headers; otherwise, your application will receive an HTTP 401 unauthorized response. An expired token is treated identically to an invalid token, so your application should watch for 401 responses to know when to use the refresh token to obtain a new access and refresh token pair. To pass the access token simply add the following header to your HTTP request:

Name: Value

Authorization: Bearer ACCESSTOKENVALUE

Refresh token usage

If your token has expired or been revoked, you may still receive an HTTP 401 unauthorized response when calling our APIs. If this happens your application should attempt to get a new access token using its refresh token and client credentials. To make this request, you will need to call the token endpoint on our identity server. Below are the details on how to make that request. The example below shows the relative path of /connect/token. The full path is found above, labeled “Token API Endpoint,” and will depend on which version of the service (e.g. development or production) you are sending requests to.

Another thing to note in this example is the expected Content-Type. Token requests must be sent as application/x-www-form-urlencoded. The fields (e.g. client_secret) must be URL/Percent Encoded, and must appear in the *body* of the request and *not* as query string parameters, even though they look similar to query string parameters.

Token Endpoint Request:

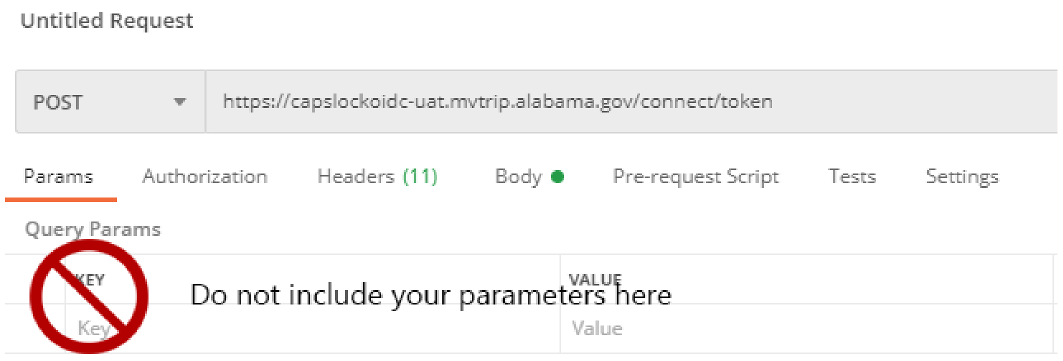
```
POST /connect/token
Content-Type: application/x-www-form-urlencoded

  client_id=YOURCLIENTID
    &client_secret=YOURCLIENTSECRET
    &grant_type=refresh_token
    &refresh_token=YOURREFRESHTOKEN
    &scope=openid%20profile
```

Token Endpoint Response:

```
HTTP/1.1 200 OK
  Content-Type: application/json
  Cache-Control: no-store
  Pragma: no-cache

  {
   "access_token": "TlBN45jURg",
   "token_type": "Bearer",
   "refresh_token": "9yNOxJtZa5",
   "expires_in": 3600
  }
```

A valid response from the token endpoint means your application should update its current access and refresh token values to the values received from the endpoint. After that has been done your application can proceed to make requests to our APIs using the newly obtained access token. Note that if a token is not refreshed more often than 30 days, a new pair cannot be obtained programmatically, and will have to come from the website just as the original pair did.

A common tool for testing REST endpoints is Postman. An example of building a request can be seen below:


Untitled Request

POST https://capslockoidc-uat.mvtrip.alabama.gov/connect/token

Params | Authorization | Headers (11) | Body ● | Pre-request Script | Tests | Settings

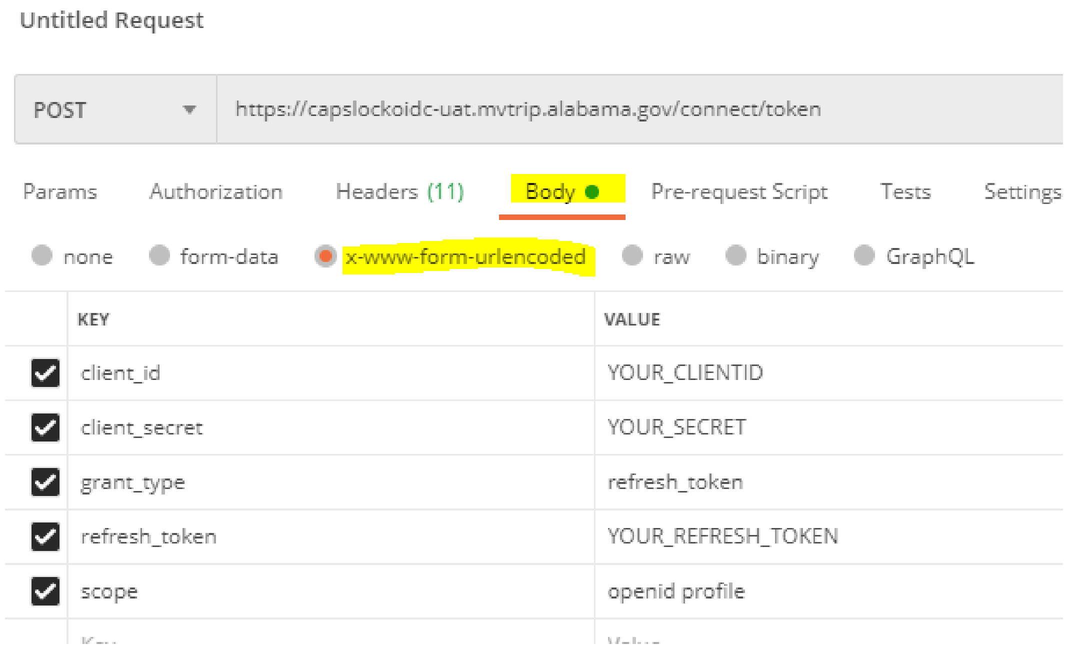
Query Params

KEY	VALUE
Key	Value

Do not include your parameters here

In the above screenshot, you should *not* include parameters as part of the query.

Instead, include them in the body, as seen below:


Untitled Request

POST https://capslockoidc-uat.mvtrip.alabama.gov/connect/token

Params | Authorization | Headers (11) | Body ● | Pre-request Script | Tests | Settings

none | form-data | ● x-www-form-urlencoded | raw | binary | GraphQL

	KEY	VALUE
✓	client_id	YOUR_CLIENTID
✓	client_secret	YOUR_SECRET
✓	grant_type	refresh_token
✓	refresh_token	YOUR_REFRESH_TOKEN
✓	scope	openid profile

Token revocation

Token revocation can happen for a number of reasons including: your token(s) have been stolen, a user is being malicious, or your application's access is revoked. If your application's tokens are revoked, but you still need access to the system, you will need to go back to the obtaining tokens setup section of this document and repeat steps 4 and 5 to obtain a new set of tokens.

Open API

The ALTS API uses OpenAPI (formerly called Swagger) to make it easier for clients to interact. More information about this project can be found at https://www.openapis.org/faq. One major benefit is that clients can be generated automatically for a variety of different languages, similar to a WSDL for older SOAP services. For more information about this, please see https://swagger.io/tools/swagger-codegen/. The API itself also exposes an OpenAPI front-end that shows a full definition of the endpoints, methods, and models. The address will be the same domain as those above, with "/swagger" at the end, so for example https://altsapi-uat.mvtrip.alabama.gov/swagger. This page will also have a link to the most current version of this document.

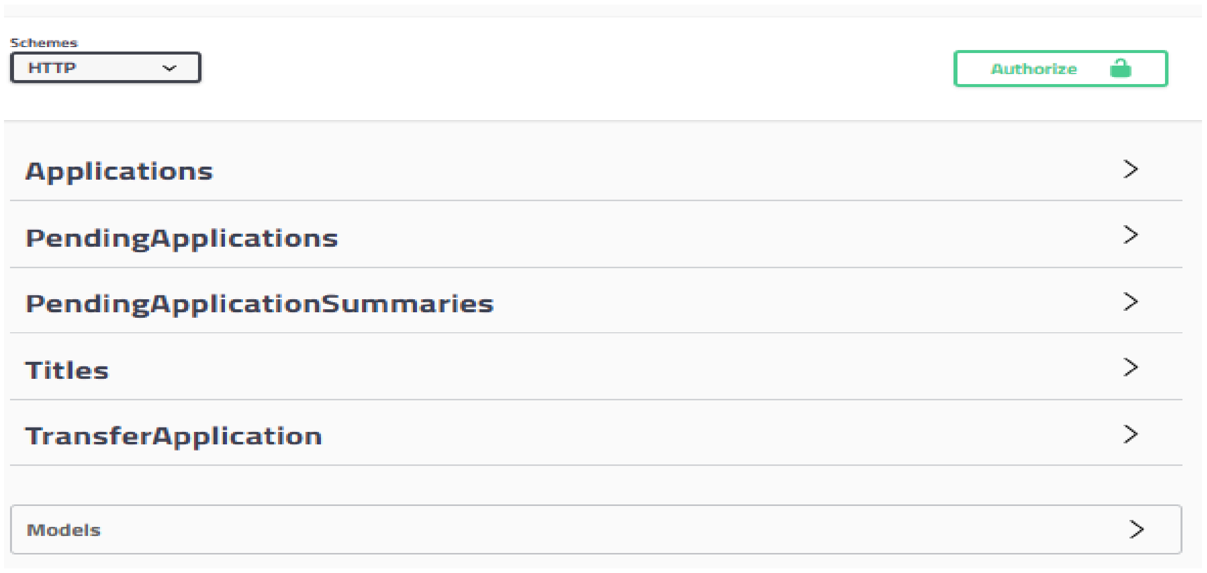


Figure 1 Operations / Endpoints

Operations

Below is a brief overview of some of the operations the API will support. Model types will appear `like this`, and in every case, the model's definition can be found on the website under the `Models` section:

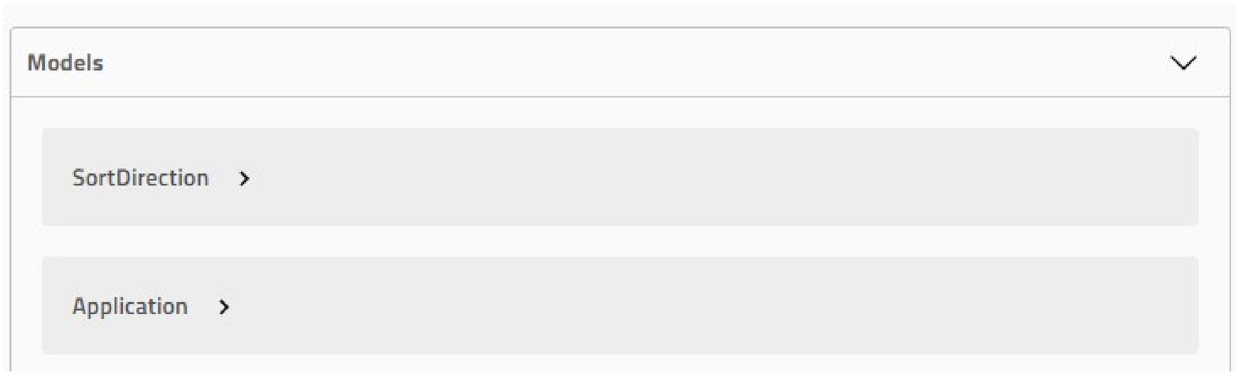
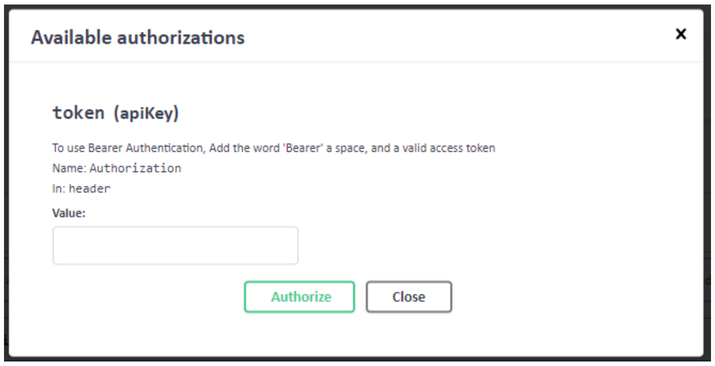


Figure 2 Models

Testing API Calls Directly in the API Explorer

It is possible to test most operations directly in the API explorer. At the top of the site there is an Authorize button. Alternatively, on each operation you will also see an open lock icon (). Clicking either of these buttons will display a box that allows you to add the access token, which will then be sent with any requests made through the site.



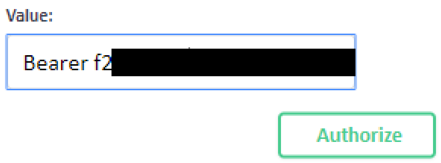
Available authorizations ×

token (apiKey)

To use Bearer Authentication, Add the word 'Bearer' a space, and a valid access token
Name: Authorization
In: header
Value:

Authorize Close

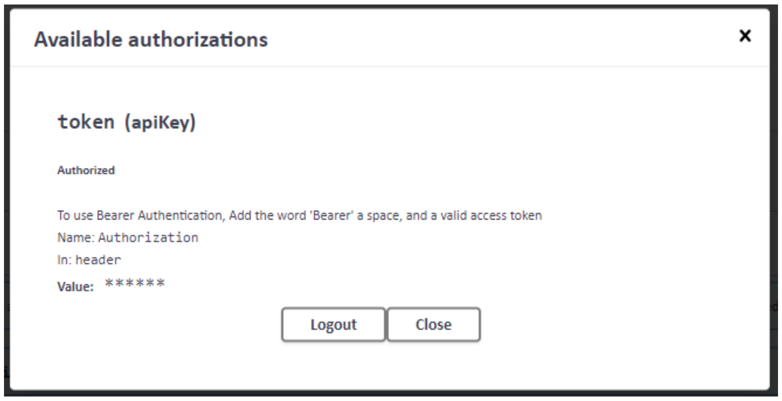
The header that will be added is the standard `Authorization` header. Similar to adding the Authorization header in any actual API requests you make, you will need to manually add “`Bearer` “ (note the space) before your access token in the test site. So, for example:



Value:

Bearer f2[redacted]

Once you click the button, you can close the Authorization dialog.



Available authorizations

token (apiKey)

Authorized

To use Bearer Authentication, Add the word 'Bearer' a space, and a valid access token

Name: `Authorization`

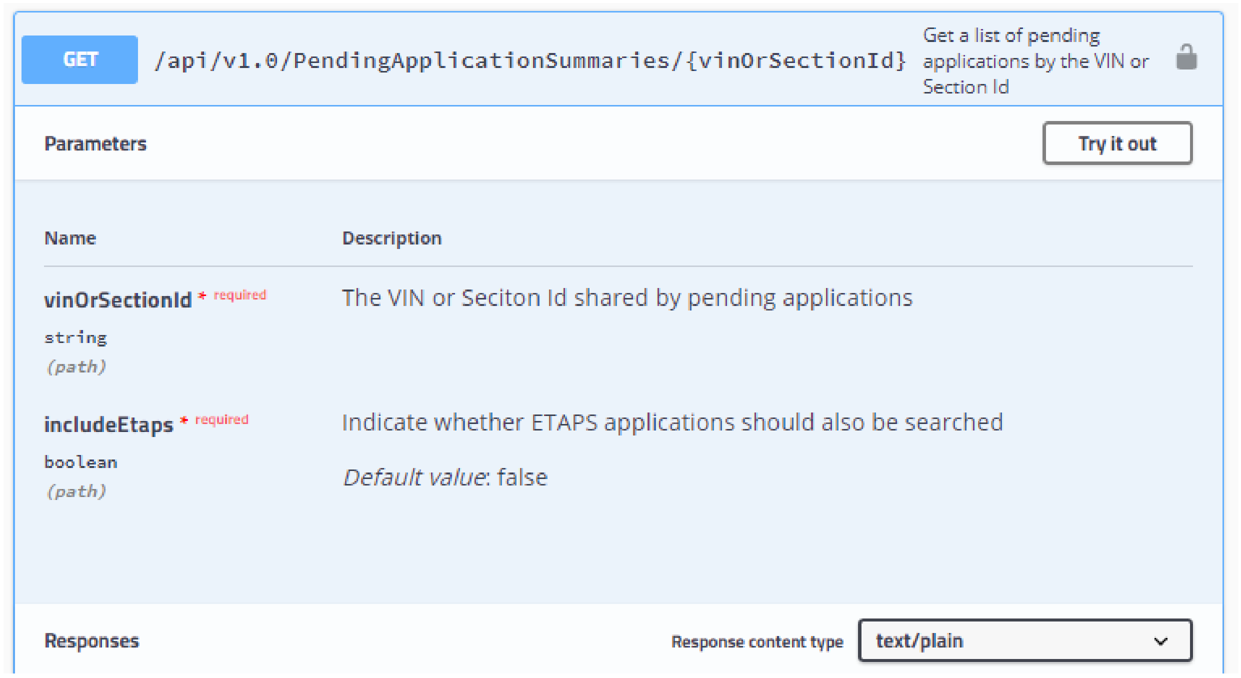
In: `header`

Value: ******

If you need to update your token or wish to clear it, simply click the button to enter a new one.

Querying for Applications/Titles

There are several options for querying for the existing title or pending application information. The specific options available on the UI can be seen in Figure 1. It will show the types expected, the return types for each one, and other relevant information. For example, if you wanted to check for pending applications for VIN 1234, send a GET request to `/api/v1.0/PendingApplicationSummaries/1234`. You can test sending requests directly from the browser by clicking the “Try it out” button.



GET `/api/v1.0/PendingApplicationSummaries/{vinOrSectionId}` Get a list of pending applications by the VIN or Section Id

Parameters

Name	Description
vinOrSectionId * required string (path)	The VIN or Seciton Id shared by pending applications
includeEtaps * required boolean (path)	Indicate whether ETAPS applications should also be searched *Default value*: false

Responses

Response content type: text/plain

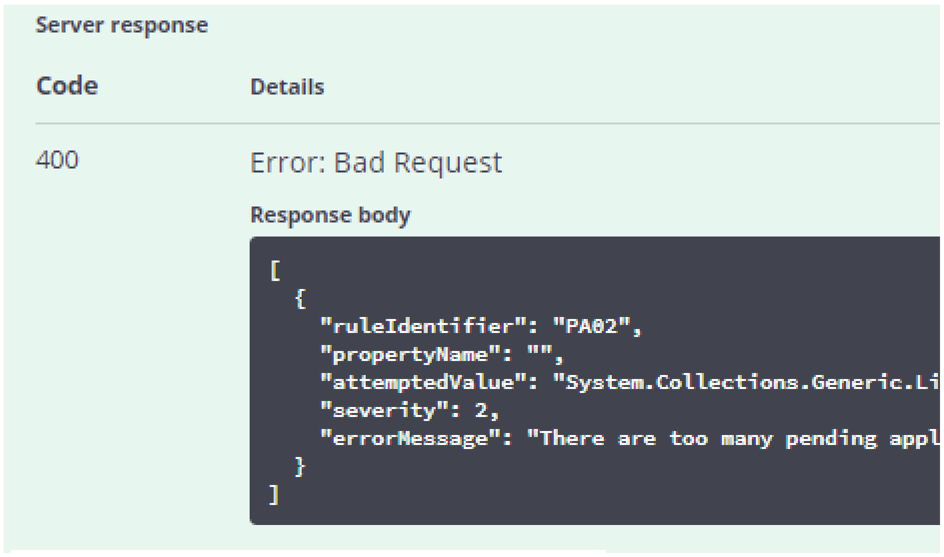
Figure 1.2 Submenu of Operations / Endpoints

Note: This is just an example. You can see the full list of operations on the website.

Creating Transfer Applications

Currently, the API only supports creating new Transfer applications for vehicles and manufactured homes. This functionality will be expanded over time. To create a new vehicle transfer application, you will post a valid `TransferApplication` object to `/api/Application/Transfer`. Likewise, to create a new manufactured home transfer application, you will post a valid `MhTransferApplication` object to `/api/Application/MhTransfer`. In either case, you will receive a `CreateApplicationResult`. If there are any validation errors that prevented it from being created or completed, it will be contained in this object. If the request is successful, the resulting object will contain summary information about the new application including the application number, and URLs for retrieving it via the API or viewing it in the ALTS website.

In the current version, to edit an existing application you must visit the ALTS website, but this will also be enhanced in the future. To have the best chance of completing an application without needing to visit the ALTS website, you can validate your potential application before attempting to create it. To do this, you must first post your `TransferApplication` to `/api/v1.0/Application/Transfer/Validate and MhTransferApplications to /api/v1.0/Application/MhTransfer/Validate`. In this case, rather than a full `CreateApplicationResult`, you will receive just the list of `ValidationFailure` objects.



```
[
  {
    "ruleIdentifier": "PA02",
    "propertyName": "",
    "attemptedValue": "System.Collections.Generic.Li
    "severity": 2,
    "errorMessage": "There are too many pending appl
  }
]
```

Figure 3 Example Response

Example Scenarios

The full `TransferApplication` model contains a large number of properties and objects. This can be a bit overwhelming, so here are a few example scenarios.

One thing that it helps to remember is that for many of the various properties, you only need include the ones that are relevant to the application you are trying to create. For example, if a party name belongs to an individual, there is no need to include `LegalBusinessName` in your `PartyName` object.

Another good example of omitting unnecessary properties is the Vehicle object itself. When querying for Applications or Titles, it is helpful to have some human-readable properties in the objects. For example, `MakeName` exists as a quick way to get the Make of the vehicle regardless of whether the Make field is one of the enumerated types (`VehicleMakeType` enumeration) or if Make has been set to `VehicleMakeType.Other`, and a value has been supplied for `MakeOtherName`. In the context of *creating* applications, however, `MakeName` is not particularly relevant, because it is not a writable field. Because it is not writable, it can be omitted when sending application objects. It does not hurt to send it, but it will be ignored. Below are a few concrete scenarios that should help illustrate this.

Scenario #1: Create an Out of State Vehicle Application

Using the following JSON request body, you can create a vehicle `Transfer` application using an Out of State title. In this example, the party names are all businesses. Because they are all of the `Business PartyNameType`, notice that the `FirstName`, `LastName`, etc. fields have been omitted. This simplifies creating the JSON payloads. The standard endpoint is `/api/v1.0/Application/Transfer`, but in addition to the body of the request, it can take one additional parameter. By default, ALTS will try to complete the new application so that it is ready to print or submit. If you would rather leave the application in `SavedInProgress` status, you may append `completeApplication=false` to the end of your request, which would become `/api/v1.0/Application/Transfer?completeApplication=false`

In this case, we'll leave it off.

`POST /api/v1.0/Application/Transfer`

```
{
  "ApplicationType": "Transfer",
  "ApplicationCategory": "Vehicle",
  "HasBrandChanges": false,
  "HasOperator": false,
  "HasSpecialMailing": false,
  "IsOwnershipChanging": true,
  "SaleInformation": {
      "DealerInventoryDate": "2018-10-25T00:00:00",
      "GrossSellingPrice": 350,
      "TradeInAllowance": 150,
      "PurchasedDate": "2018-11-25T00:00:00"
  },
  "Seller": {
      "SellerTaxIdNumber": "ABC123",
      "SellerLicenseNumber": "492349",
      "SellerLicense": "AutomobileLicense",
      "Names": [
          {
              "PartyNameType": "Business",
              "LegalBusinessName": "Some Business Inc."
          }
      ],
      "Address": {
         "StreetAddress": "123 Street",
         "City": "Tuscaloosa",
         "State": "Tennessee",
         "ZipCode": "34952",
         "Country": "Usa"
     }
  },
  "Vehicle": {
      "BodyStyle": "CarryAll",
      "Code": "Used",
      "Color1": "Black",
      "Color2": "Unknown",
      "FuelType": "Gas",
      "GvwrCode": "ZeroTo6K",
      "IsTrailer": false,
      "Make": "Jeep",
      "MakeName": "Jeep",
      "MakeOtherName": "",
      "Model": "GRAND CHEROKEE",
      "NumberOfCylinders": 4,
      "OdometerReading": null,
      "OdometerReadingType": "Exempt",
      "OdometerUnits": "Miles",
      "Vin": "1J4GZ58Y0TC271410",
      "Year": 1996
  },
  "PrimaryDocumentType": "OutOfStateTitle",
  "PrimaryDocument": {
      "IsSurrenderedToAl": true,
      "IsTitleUnderBond": false,
      "TitleNumber": "MS3949243",
      "TitleState": "Tennessee",
      "DocumentType": "OutOfStateTitle",
      "IssueDate": "2018-04-23"
  },
  "IsVehicleLeased": false,
  "Owner": {
      "IsLessor": false,
      "Email": null,
      "Phone": null,
      "Names": [
          {
              "PartyNameType": "Business",
              "LegalBusinessName": "Some Business Inc."
          }
      ],
      "Address": {
         "StreetAddress": "123 Street",
         "City": "Tuscaloosa",
         "State": "Alabama",
         "ZipCode": "35401",
         "Country": "Usa"
     }
  },
  "Lienholders": [
      {
         "LienDate": "2018-11-25T00:00:00",
         "Email": null,
         "Address": {
            "StreetAddress": "234 Street Avenue",
            "City": "Tuscaloosa",
            "State": "Alabama",
            "ZipCode": "35401",
            "Country": "Usa"
         },
         "Names": [
            {
               "PartyNameType": "Business",
               "LegalBusinessName": "Some Lienholder"
            }
         ]
      }
  ]
}
```

If this request is successful, you should receive something similar to this

```
Result: 201 (Created)

{
   "applicationType": "Transfer",
   "applicationStatus": "Completed",
   "resultStatus": "ApplicationCompleted",
   "applicationNumber": "TRTL10000056301",
   "applicationWebsiteUrl": "https://alts-uat.mvtrip.alabama.gov/Application/Transfer/Summary?applicationid=10000056301",
   "applicationApiUrl": "https://altsapi-uat.mvtrip.alabama.gov/api/Applications/TRTL10000056301",
   "validationFailures": []
 }
```

The `201` response status code indicates that the application has been *created,* but you will need to check the `ResultStatus` field to determine the state of the application. If the application still has validation errors, the `ResultStatus` field will be `ValidationFailure`. In this case, the user will need to go to the URL in the `ApplicationWebsiteUrl` field to fix any problematic data. Currently updating applications is not supported via the API, so this must be done on the ALTS website.

If the request has errors that prevent the application from being saved (too many pending applications for that VIN, for example), you will receive a 400 response status code and the resulting object will not contain `ApplicationWebsiteUrl` or

`ApplicationApiUrls`. It will contain validation errors something like the below, however. In this case, no application has been created, so you are free to resolve any errors returned and try again.

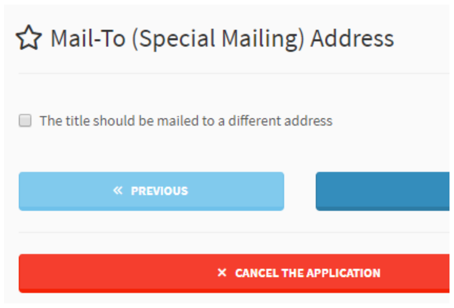
```
Result: 400 (Bad Request)

 {
   "applicationType": "Transfer",
   "applicationStatus": "Unknown",
   "resultStatus": "ValidationFailure",
   "validationFailures": [
     {
       "ruleIdentifier": "PA02",
       "propertyName": "",
       "attemptedValue": "System.Collections.Generic.List`1[Alts.Logic.AdminWorkflow.PendingApplicationSummary]",
       "severity": "Error",
       "errorMessage": "There are too many pending applications for this VIN. Existing application numbers: TRTL10000014501"
     }
   ]
 }
```

One thing that may stick out in that example is the apparent redundancy between `PrimaryDocumentType` and the `DocumentType` field of `PrimaryDocument`. If we were concerned *only* with vehicle applications, this would obviously be unnecessary. The reason relates to the structure of manufactured homes, which can have a primary document *per* section. Currently each manufactured home section must have the same document type (e.g. an out of state title for each one), but the model is built to accommodate potential changes to that rule in the future.

One other interesting thing about this example is the presence of a couple of the "`Has…`" fields. These correspond directly to choices made in the existing ALTS UI. For example, `HasSpecialMailing` indicates to the application whether it should attempt to save a `SpecialMailing` field. In the example above, this field is set to `false`, and so the `SpecialMailing` property is omitted. Also because it is set to `false`, even if it were included, it would be ignored. However, if `HasSpecialMailing` were set to `true`, the API would expect that object to be included. Including that might then look something like this:

```
"HasSpecialMailing": true,
"SpecialMailing":{
 "Names": [
    {
      "PartyNameType": "Business",
      "LegalBusinessName": "Some Business Inc."
    }
 ],
 "Address": {
   "StreetAddress": "123 Street",
   "City": "Tuscaloosa",
   "State": "Tennessee",
   "ZipCode": "34952",
   "Country": "Usa"
 }
```

These properties are the same as used in the ALTS website. Here is `HasSpecialMailing` when viewed in the context of the UI.


Mail-To (Special Mailing) Address

The title should be mailed to a different address

Scenario #2: Create an In-State Vehicle Transfer Application

Using the following JSON request body, you can create a Transfer application using an existing Alabama title. Note that because only one valid Transfer application is allowed for a VIN at any given time, this exact VIN/title number combination likely will not actually work on the test site. The only real change you would have to make for testing purposes would be to change the previous title information. Existing Alabama titles that work with the testing site can be located at https://alts-uat.mvtrip.alabama.gov. The same credentials used to obtain access tokens can be used to log into this test version of ALTS.

You will probably notice that the test data below is very similar to the JSON body in Scenario #1. In fact, the only real substantive difference in this scenario is the `PrimaryDocumentType` and corresponding `PrimaryDocument` object. Because our `PrimaryDocumentType` (`SupportingDocumentType` enumeration) is `AlabamaTitle` in this case, the fields that `PrimaryDocument` contain will be different from Scenario #1. Only fields relevant to a `PreviousAlabamaTitle` should be used instead of those for `OutOfStateTitle` in Scenario #1.

```
POST /api/v1.0/Application/Transfer
```

```
{
  "ApplicationType": "Transfer",
  "ApplicationCategory": "Vehicle",
  "HasOperator": false,
  "HasSpecialMailing": false,
  "IsOwnershipChanging": true,
  "SaleInformation": {
      "DealerInventoryDate": "2018-10-25T00:00:00",
      "GrossSellingPrice": 350,
      "TradeInAllowance": 150,
      "PurchasedDate": "2018-11-25T00:00:00"
  },
  "Seller": {
      "SellerTaxIdNumber": "ABC123",
      "SellerLicenseNumber": "492349",
      "SellerLicense": "AutomobileLicense",
      "Names": [
          {
              "PartyNameType": "Business",
              "LegalBusinessName": "Some Business Inc."
          }
      ],
      "Address": {
         "StreetAddress": "123 Street",
         "City": "Tuscaloosa",
         "State": "Tennessee",
         "ZipCode": "34952",
         "Country": "Usa"
     }
  },
  "Vehicle": {
      "BodyStyle": "CarryAll",
      "Code": "Used",
      "Color1": "Black",
      "Color2": "Unknown",
      "FuelType": "Gas",
      "GvwrCode": "ZeroTo6K",
      "IsTrailer": false,
      "Make": "Jeep",
      "MakeName": "Jeep",
      "MakeOtherName": "",
      "Model": "GRAND CHEROKEE",
      "NumberOfCylinders": 4,
      "OdometerReading": null,
      "OdometerReadingType": "Exempt",
      "OdometerUnits": "Miles",
      "Vin": "1GNES16S326115264",
      "Year": 1996
  },
  "PrimaryDocumentType": "AlabamaTitle",
  "PrimaryDocument": {
      "DocumentType": " AlabamaTitle",
      "TitleNumber": "51704160",
      "IssueDate": "2018-04-23"
  },
  "IsVehicleLeased": false,
  "Owner": {
      "IsLessor": false,
      "Email": null,
      "Phone": null,
      "Names": [
          {
              "PartyNameType": "Business",
              "LegalBusinessName": "Some Business Inc."
          }
      ],
      "Address": {
         "StreetAddress": "123 Street",
         "City": "Tuscaloosa",
         "State": "Alabama",
         "ZipCode": "35401",
         "Country": "Usa"
     }
  }
}
```

The return types in Scenario #2 will be the same as in Scenario #1, with a successfully created application returning various information about the new application, or a failure returning the validation errors.

Scenario #3 Create an Out of State Manufactured Home Application

Creating manufactured home applications is very similar to creating vehicle applications. The first and main difference is that instead of a `Vehicle` object, you have a `ManufacturedHome` object in its place. Because manufactured homes can have up to four sections (sometimes called "sides"), there may be multiple identifiers. For vehicles, the identifier is the VIN (Vehicle Identification Number). A similar standard does not exist for manufactured homes, and so the identifier is simply called `SectionId`. You may see this referenced as a VIN as well, though that is more by convention, and is not entirely accurate.

The second major difference is that instead of a single `PrimaryDocument` object at the Application level, each manufactured home section has its own `PrimaryDocument`. It is normal to have only a single section, but it is possible to have as many as four. The root Application object does still have a `PrimaryDocumentType`, as currently each side's document type is required to be the same.

Using the following JSON request body, you can create a manufactured home `MhTransfer` application

```
POST /api/v1.0/Application/MhTransfer
```

```
{
   "ApplicationType": "MhTransfer",
   "ApplicationCategory": "ManufacturedHome",
   "HasOperator": false,
   "HasSpecialMailing": false,
   "IsOwnershipChanging": true,
   "SaleInformation": {
      "DealerInventoryDate": "2018-10-25T00:00:00",
      "GrossSellingPrice": 350,
      "TradeInAllowance": 150,
      "PurchasedDate": "2018-11-25T00:00:00"
   },
   "Seller": {
      "SellerTaxIdNumber": "ABC123",
      "SellerLicenseNumber": "492349",
      "SellerLicense": "AutomobileLicense",
      "Names": [
         {
            "PartyNameType": "Business",
            "LegalBusinessName": "Some Business Inc."
         }
      ],
      "Address": {
         "StreetAddress": "123 Street",
         "City": "Tuscaloosa",
         "State": "Tennessee",
         "ZipCode": "34952",
         "Country": "Usa"
      }
   },
   "PrimaryDocumentType": "OutOfStateTitle",
   "ManufacturedHome": {
      "Sections": [
         {
            "SectionId": "6516516516516541",
            "PrimaryDocument": {
               "IsSurrenderedToAl": true,
               "IsTitleUnderBond": false,
               "TitleNumber": "MS3949243",
               "TitleState": "Tennessee",
               "DocumentType": "OutOfStateTitle",
               "IssueDate": "2018-04-23"
            }
         }
      ],
      "Color": "White"
      "Make": "Mfg Co",
      "Model": "Home",
      "Year": 2008
   },
   "Owner": {
      "IsLessor": false,
      "Email": null,
      "Phone": null,
      "Names": [
         {
            "PartyNameType": "Business",
            "LegalBusinessName": "Some Business Inc."
         }
      ],
      "Address": {
         "StreetAddress": "123 Street",
         "City": "Tuscaloosa",
         "State": "Alabama",
         "ZipCode": "35401",
         "Country": "Usa"
      }
   }
}
```

Your response types will be the same as those from Scenario #1. The potential validation codes and errors may be different, since the rules are specific to Manufactured Home applications, but the structure of the responses should be the same.