RESTful API v2

May 2016

Contents

Introduction

Pay Advantage provides an API for simpler integration into your existing systems and applications; perfect for clients who need to automate the process of direct debiting or automate their customer and BPAY facilities.

The API has been written as a RESTful API and classes can be accessed using standard Post, Get, Put, and Delete actions. Responses are returned using JSON and examples of responses are included for easy reference.

You will require a good understanding of web services, JSON, and REST to utilise this API.

Configuration

Pay Advantage API can be accessed using the following URI:
https://api.payadvantage.com.au/v2

The test API can be accessed using the following URI:
https://api.test.payadvantage.com.au/v2

Your user credentials are accessed from your online Pay Advantage account. Click on the “API” link under the “Account & Settings” section in the settings menu (top right). You need administrator privileges to do this.

Authorisation to use the API features is controlled via registered IP addresses. **You need to add any IP Addresses that will be used to access the API** in this section.

Authorisation and Authentication

A call is made to the endpoint below with form variables “username=[your_username]”, “password=[your_password]” and “grant_type=password". Here is an example of a typical request:

```
POST https://api.test.payadvantage.com.au/V2/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: api.test.payadvantage.com.au
Content-Length: 94
Expect: 100-continue
Connection: Keep-Alive

grant_type=password&username=506530e9283d7w0e4i7b4t783&password=038264873276ns7980427js4f40611
```

For a successful authentication the result is a JSON object of the form below

```
{
   "access_token":"token string contents",
   "token_type":"bearer",
   "expires_in":599
}
```

For an unsuccessful authentication the result is a JSON object of the form below

```
{"error":"unsupported_grant_type"}
```

The authorization token returned from a successful authentication must be included in any subsequent requests as an authorisation header called “Bearer”. Authentication tokens have an expiry (in seconds) which is returned in the “expires_in” field as shown above. After this time any request to the API with an expired token will return an unauthorized response (401) and a new token will need to be requested and appended to new requests.

```
<html>
  <body>
    <form action="https://api.payadvantage.com.au/v2/token" method="post">
      <input name="grant_type" value="password" type="hidden" />
      <input name="username" value="ENTER USERNAME" type="hidden" />
      <input name="password" value="ENTER PASSWORD" type="hidden" />
      <input type="submit" value="Submit">
  </body>
</html>
```

Restrictions, Performance & Best Practice

We aim to make our API as fast and accessible for our users as possible. Due to the fact that we have no real control over how you choose to make use of the API we have put in place appropriate restrictions to prevent inefficient and redundant requests to our API. The following items are returned in each response’s headers.

X-Rate-Limit-Limit - The number of requests allowed in the current period
X-Rate-Limit-Remaining - The number of requests remaining in the current period
X-Rate-Limit-Reset - The number of seconds left in the current period

Please refer to the online help article for a comprehensive list of limit values, best practice, performance tips and recommended batch processing times:
https://help.payadvantage.com.au/hc/en-us/articles/218958268

List Methods

To get started it is easy to simply get lists of data. We have listed all of the available “list” methods that you can call below. Please note that you may append additional parameters to many of these calls as detailed later in this document.

- `GET: /customers`
- `GET: /payments`

- `GET: /payments_settled/{ddMMyyyy}` – *date is optional, defaults to today. Can still apply all parameters below.*
- `GET: /payments_failed/{ddMMyyyy}` – *date is optional, defaults to today. Can still apply all parameters below.*
- `GET: /debit_instructions`
- `GET: /debit_instructions_for_batch/{batch_code}` – *batch_code is **required**, adds this filter. Can still apply all parameters below.*
- `GET: /debit_instructions_failed/{ddMMyyyy}` – *date is optional, defaults to today. Can still apply all parameters below.*
- `GET: /debit_batches`
- `GET: /debit_batches/{code}`
- `GET: /debit_batches/{code}/debit_instructions`

List Result Format

List requests return two main properties within the response object – Records and Meta. "Records" is an array of the items in the list and "Meta" contains paging information as well as helper links for you to use.

```
{
  "Records": [{…},{…},{…}],
  "Meta":
  {
    "page": 3,
    "recs_per_page": 50,
    "total_recs": 412
  }
}
```

List Parameters (optional)

You can customise the list data returned using the following query parameters. These parameters follow the same rules as any standard HTML query string parameter. The first parameter is separated by '?' and subsequent parameters are separated by '&'.

Parameter	*Description*
page={x}	{x} is the zero-based index of the page of records to return. Page indexes greater than last page will return an empty page. Negative page indexes will return an error.
per_page={x}	{x} represents records per page. Maximum value is 1000, default value is 100. Values greater than maximum or negative values will return an error.
sort={x,-y,z}	Allows you to sort lists by a defined set of fields. A minus (-) symbol in front of the field indicates descending order. The full list of **sortable** fields can be found under the heading for each class. Field names are **case insensitive** and don't contain spaces. Invalid field names will return an error.
fields={x,y,z,…}	You can specify a comma-separated list of the field names to include for any list. This helps streamline your data, reduce transfers, and enable easier parsing. The order of these fields is respected in the returned result objects. You can see a full field name listing under the heading for each class. Field names are **case insensitive** and don't contain spaces. Invalid field names will return an error.
Include={x,y,z}	Some fields are not included in list results by default. You can add them here if you would like them included. EmbeddedObjects (identifiable in the fields lists with a field name containing a ".") can have

	their fields explicitly included by using the full dotted field name for example include=createdby.firstname”
exclude={x,y,z}	You can specify a comma-separated list of the field names to exclude for any list. This helps streamline your data, reduce transfers, and enable easier parsing. You can see a full field name listing under the heading for each class. Field names are **case insensitive** and don’t contain spaces. Invalid field names will return an error.
{fieldname}={value}	You can filter on field values using this approach. The full list of **filterable** fields can be found under the heading of each class. String fields are always partial matches matching on any record the field’s string value containing the value provided. Field names are **case insensitive** and don’t contain spaces. Invalid field names will return an error. A minimum of 2 characters are required for value.
{fieldname}from={value}	Some filterable fields (typically numeric or date fields) allow a ranged search (identified by RNG in fields list). Appending “from” to the fieldname will allow for a search for all values greater than or equal to the “from” value. Field names are **case insensitive** and don’t contain spaces. Invalid field names will return an error.
{fieldname}to={value}	Some filterable fields (typically numeric or date fields) allow a ranged search (identified by RNG in fields list). Appending “to” to the fieldname will allow for a search for all values less than or equal to the “to” value. Field names are **case insensitive** and don’t contain spaces. Invalid field names will return an error.

List Examples

Retrieve the 4th page of customers

`/customers?page=3`

Retrieve the 4th page of customers with 500 records per page

`/customers?page=3&per_page=500`

Retrieve the 4th page of customers with 500 records per page, but only Code, Name, BillerCode, and BPAYRef fields

`/customers?page=3&per_page=500&fields=code,name,billercode,bpayref`

Individual Record Methods

To retrieve more information on a particular object you can call the following methods, supplying the identifier for the desired object in '{code}'.

- `GET: /customers/{code},{code}` *– Separate code using comma, max 1000 codes*
- `GET: /payments/{code}`
- `GET: /debit_batches/{code}`
- `GET: /debit_instructions/{code}`

Individual Record Result Format

All of these methods return an individual JSON object matching the identifier (code) provided. An example of a typical response is shown below.

```
{
  "Code": "88V3FE",
  "StringField": "Some value",
  "IntField": 1234,
  "DateField": "yyyy-MM-ddT00:00:00",
  "DateTimeField": "yyyy-MM-ddTHH:mm:ss",
  "DecimalField": 12.34,
  "BooleanField": true,
  "NullableField": null,
  "EmbeddedObject": {…}
}
```

Create Methods

Create methods are further defined under the heading of each class. There may be restrictions and limitations on each class that you need to be aware of. Our API allows for the creation of the following objects:

- `POST: /customers`
- `POST: /customers?with=bpayref` *- Create a Customer Reference Number as well.*
- `POST: /debit_instructions`
- `POST: /debit_batches`

Create Result Format

On successful creation the code will be returned in the response message as below:

```
{"Code": "88V3FE"}
```

Update Methods

Update methods are further defined under the heading of each class. There may be restrictions and limitations on each class that you need to be aware of. On a successful update an individual record of the new item will be returned.

- `POST: /customers/{code}`
- `POST: /debit_instructions/{code}`
- `POST: /debit_batches/{code}`
- `POST: /debit_batches/{code}/confirm`
- `POST: /debit_batches/{code}/authorise`

Delete Methods

Delete methods are further defined under the heading of each class. There may be restrictions and limitations on each class that you need to be aware of.

- `DELETE: /debit_batches/{code}`
- `DELETE: /debit_instructions/{code},{code}` *– Separate code using comma, max 1000 codes*

Customers

You can manage all of your customers, including associated payments and direct debits, using the following methods shown in this section.

Customer Record (GET: /customers/{code})

An example of a customer record is shown below. This is the record that is returned on request of a single record and an array of these records is returned when requesting lists. Greyed fields are fields not readily displayed on Pay Advantage screens, but still useful and necessary information such as identifiers and calculated fields.

```
{
  "Code": "our system id",
  "ExternalID": "your system id",
  "DateCreated": "2014-04-23T13:45:31",
  "Name": "Customer 1",
  "CustomRef": "your cust ref",
  "Email": "customer@email.com",
  "BillerCode": "012345",
  "BPAYRef": "987654321",
  "CreatedBy": { "FirstName": "John","LastName": "Smith","UserName": "js@email.com.au" }
  "URI": "/customers/{code}"
}
```

Lists of Customers (GET: /customers)

Customers are sorted by name (ascending) by default. The list is paged and will always return the first page of records unless paging parameters are added as described previously. An example of the response is shown below

```
{
  "Records":
  [
    {
      "Code": "ABC123",
      "ExternalID": "43",
      "DateCreated": "2011-01-23T16:12:41.263",
      "Name": "Test Customer",
      "CustomRef": "TST101",
      "Email": "test@email.com.au",
      "BillerCode": "212886",
      "BPAYRef": "10000111111"
    },
    {…}
  ],
  “Meta”: { "page": 0, "recs_per_page": 100, "total_recs": 500 }
}
```

Creating a Customer (POST: /customers)

To create a new Customer you simply need to post to the address above. “Name” is the only mandatory field and you can provide additional customer fields if they are to be populated. Editable fields can be identified by reviewing the tables at the end of this document.

```
{
  "Name":”Bob Smith”,
  “Email”:”bob@yahoo.com”
}
```

Updating a Customer (POST: /customers/{code})

To update a Customer you need to post to the address above and include the identifier for the Customer in “{code}”. You only need to provide Customer fields that are being updated. The example below is updating the email address for customer with ID of ABCD123.

```
POST: /customers/ABCD123
{
  “Email”:”bob@live.com”
}
```

Debit Batches

Debit Batches are used to batch many debit instructions together ─ consolidating the management of individual Debit Instructions. Please refer to the online help article for more information relating to Batch Debiting:

https://help.payadvantage.com.au/hc/en-us/articles/203572406

Debit Batch Record (GET: /debit_batches/{code})

An example of a Debit Batch record is shown below. This is the record that is returned on request of a single record and an array of these records is returned when requesting lists. Greyed fields are fields not readily displayed on Pay Advantage screens, but still useful and necessary information such as identifiers and calculated fields.

```
{
  "Code": "our system id",
  "DateCreated": "2014-04-23T13:45:31",
  "Name": "Your file name or ref",
  "DateToDebit": "2014-04-25T00:00:00",
  "IsConfirmed": true,
  "IsAuthorised": true,
  "IsProcessed": true,
  "RemitterName": "XYZ Trading",
  "DebitInstructionCount": 87,
  "DebitInstructionAmountSum": 4985.72,
  "CreatedBy": { "FirstName": "John","LastName": "Smith","UserName": "js@email.com.au" }
  "URI": "/debit_batches/{code}"
}
```

Lists of Debit Batches (GET: /debit_batches)

Debit Batches are sorted by DateCreated (descending) by default. The list is paged and will always return the first page of records unless paging parameters are added. An example of the response is shown below.

```
{
  "Records":
  [
    {
      "Code": "888ABC",
      "DateCreated": "2014-04-15T09:12:30.983",
      "Name": "Low Value Test",
      "DateToDebit": "2014-04-20T00:00:00",
      "IsConfirmed": true,
      "IsAuthorised": true,
      "IsProcessed": true,
      "RemitterName": "PAY ADVANTAGE",
      "DebitInstructionCount": 1,
      "DebitInstructionAmountSum": 1.00
    },
    {…}
  ],
  "Meta": { "page": 0, "recs_per_page": 100, "total_recs": 500 }
}
```

Creating Debit Batches (POST: /debit_batches)

The following example illustrates how to create a new Debit Batch. “Name” and “DateToDebit” are mandatory. You can include an initial array of DebitInstructions or send an empty “DebitInstructions” array and add them later. Omitting the “RemitterName” will default to the remitter name configured for your Pay Advantage account.

```
{
  “Name”: “A mandatory field”,
  “DateToDebit”: “2014-04-27”,
  “RemitterName”: null,
  “DebitInstructions”: [{DebitInstructionRecord}, {…}]
}
```

Updating Debit Batches (POST: /debit_batches/{code})

The following example illustrates how to update an existing Debit Batch. You only need to include the fields you are updating.

```
{
  “DateToDebit”: “2014-04-25”,
  “RemitterName”: “ABC Training”
}
```

Confirming, Authorising and Cancelling Debit Batches

Once a debit batch has been created and all Debit Instructions added, it must be confirmed and authorised. The permissions system allows one level of user to confirm Debit Batches are correct and a higher level user gives the authority to proceed with the debits. Depending on your organisation’s requirements, you can bypass the confirmation stage via this API and confirm and authorise in one step by jumping straight to the “/authorise” call. Debit Batches can also be cancelled at any time before being processed.

```
PUT: /debit_batches/{code}/confirm
PUT: /debit_batches/{code}/authorise
PUT: /debit_batches/{code}/cancel
```

Deleting a Debit Batch

A debit batch can be deleted provided it is not in a confirmed or authorised state. This deletes all instructions within it.

```
DELETE: /debit_batches/{code}
```

Debit Instructions

Debit Instructions are individual instructions to debit an amount from a bank account. We record the success or failure of each instruction and nothing more.

On-charging Fees

If you decide to on-charge debit fees directly to your customers then you can make use of the OnchargeFees field to simplify your processing effort and let us manage this for you. Debit Fees will be calculated when the Debit Instruction is processed for payment (typically the DateToDebit from the DebitBatch). The fee will be added to the payment at this point for on-charged debit instructions.

Group Key Feature

Any Debit Instruction may have a Group Key assigned. As the name suggests, this field is used for grouping related instructions together. The same group key may be used across any number of batches over any period of time. Please be aware that, when using the Group Key and On-charge features together, any on-charged fee that dishonours will be added to the next on-charged debit instruction with a matching group key. This is by design to assist in fee management and recovery. It is only a real concern if on-charging is used and the group key is shared between different customers. One solution in this scenario is to add a unique suffix to group key for each customer in that group (for example “AB001-01”, “AB001-02”) so these can be filtered at a later date using the first n characters.

Debit Instruction Record (GET: /debit_instructions/{code})

An example of a Debit Instruction record is shown below. This is the record that is returned on request of a single record and an array of these records is returned when requesting lists. Greyed fields are fields not readily displayed on Pay Advantage screens, but still useful and necessary information such as identifiers and calculated fields

```
{
  "Code": "our system id",
  "DateCreated": "2014-04-23T11:07:13.322",
  "IsProcessed": true,
  "BSBNumber": "012345",
  "AccountNumber": "111111111",
  "AccountName": "Test Account",
  "Amount": 87.50
  "RemitterName": "XYZ Trading",
  "OnchargeFees": true,
  "IsFailAcknowledged": false,
  "DebitBatch":
  {
    "Code": "our system id",
    "DateToDebit": "2014-04-25T00:00:00"
  },
  "Payment":
  {
    "Code": "our system id",
    "DatePaid": "2014-04-26T00:00:00",
    "DateFailed": null,
    "Amount": 88.45
  }
  "URI": "/debit_instructions/{code}"
}
```

Lists of Debit Instruction (GET: /debit_instructions)

Debit Instructions are sorted by DateCreated (descending) by default. The list is paged and will always return the first page of records unless paging parameters are added as described previously. An example of the response is shown below

Creating Debit Instructions (POST: /debit_instructions)

You can create Debit Instructions one at a time or many at once (up to 1000 at a time). The recommended (most common) way to create debit instructions is many at a time (a batch) as shown below.

```
{
  “DebitBatch”:
  {
    “Code”: “id of debit batch to add instructions to”
  },
  “DebitInstructions”:
  [
    {
      “BSBNumber”: “012345”,
      “AccountNumber”: “67890”,
      “AccountName”: “Test Account”,
      “Amount”: 123.45,
      “OnchargeFees”: true
    },
    {
      “BSBNumber”: “098765”,
      “AccountNumber”: “43210”,
      “AccountName”: “Another Account”,
      “Amount”: 85.45,
      “OnchargeFees”: false
    },
    { … }, { … }, { … }
  ]
}
```

To add an individual debit instruction, you still use the method above with only a single item listed in the “DebitInstructions” array.

Deleting a Debit Instruction (DELETE: /debit_instructions/{codes})

A debit instruction can be deleted as long as the “DebitBatch” it belongs to is not confirmed, authorised, or processed. Use the following syntax replacing {codes} with 1 to 1000 debit instruction codes (comma or semi-colon separated):

```
DELETE: /debit_instructions/{codes}
```

Managing Failed Debit Instructions

To simplify the management and handling of debit instructions Pay Advantage provides a helper method to quickly retrieve and "acknowledge" debit instruction failures. To allow for simplified processing and reconciliation, acknowledged failures will be omitted from results.

Failed instructions not Acknowledged (all) *– Recommended Method*

```
GET: /debit_instructions_failed
```

Failed instructions not Acknowledged (for specific date)

```
GET: /debit_instructions_failed/ddMMyyyy
```

If retrieving using todays date, be aware that instructions can fail at any time during the day and it is best to retrieve these at least one day in arrears or not specify a date at all.

If required you can create more complex filters using the fields available for filtering on debit instructions.

For Example:

Failed debit instructions for a week that have not been acknowledged

```
GET:/debit_instructions?isfailacknowledged=false&payment.datefailedfrom=20042014&payment.datefailedto=26042014
```

Failed debit instructions for a batch after a date that may or may not have been acknowledged

```
GET: /debit_instructions?debitbatch.code=xyz&payment.datefailedfrom=27042014
```

Acknowledging Failures

The API contains a feature to mark failures as acknowledged. This provides a fast method for identification of which records have been dealt with. To acknowledge failed debit instructions you can call the following method:

```
PUT: /debit_instructions/{code},{code}/ackfail
No content required
```
– Separate code using comma, max 1000 codes

Payments

Payments are associated with Customers (BPAY and Direct Debit) as well as Debit Instructions. BPAY payments settle immediately whereas Direct Debits and Debit Instructions can take a few days to settle. All payment types can fail for various reasons in which case “DateFailed” and “FailCode” will be populated. On occasion a payment may fail after settlement.

Payment Record (GET: /payments/{code})

An example of a payment record is shown below. This is the record that is returned on request of a single record and an array of these records is returned when requesting lists. Greyed fields are fields not readily displayed on Pay Advantage screens, but still useful and necessary information such as identifiers and calculated fields.

```
{
  “Code”: “our system id”,
  “DatePaid”:” 2014-04-20T00:00:00”
  “DateFailed”: null
  “FailCode”: null
  “FailReason”: null
  “Amount”: 100.88,
  “AmountExFees”: 100.00,
  "PaymentTypeID":8,
  "ExternalReference":”ABCD1234”,
  "BPAYReference":null,
  “URI”: “/payments/{code}”
}
```

Lists of Payment records (GET: /payments)

Payments are sorted by Date Paid (descending) by default. The list is paged and will always return the first page of records unless paging parameters are added as described previously. An example of the response is shown below

```
{
  "Records":
  [
    {
      "PaymentTypeID":8,
      “AmountExFees”: 100.00,
      “Code”: “our system id”,
      “DatePaid”:” 2014-04-20T00:00:00”
      “DateFailed”: null
      “DateClears”: “2014-04-25T00:00:00”,
      “FailReason”: null
      “Amount”: 100.88
    },
    {…}
  ],
  "Meta": { "page":0, "recs_per_page": 100, "total_recs":594 }
}
```

Managing Failed Payments

You can get the list of failed payments for a specific date using the following method:

```
GET: /payments_failed/{ddMMyyyy}
```

Avoid using todays date as payments can fail at any time during the day and it is best to retrieve these at least one day in arrears.

If required you can create more complex filters using the fields available for filtering on payments.

For example:

Failed payments for a week.

```
GET: /payments_failed?datefailedfrom=20042014&datefailedto=26042014
```

Failed payments after a date of more than a specific amount.

```
GET: /payments_failed?datefailedfrom=20042014&amountfrom=100
```

Managing Settled Payments

You can get the list of settled payments for today (or any day) using the following methods:

Today

```
GET: /payments_settled
```

Any day

```
GET: /payments_settled/ddMMyyyy
```

If using the “today” method, it is recommended to run this as late in the day as possible (6pm - Midnight) so that any settlements that occurred during the day are captured. You can also add more complex filters if needed using the fields available for filtering on debit instructions. See some examples below:

Settled payments for a week.

```
GET: /payments_settled?datesettledfrom=20042014&datesettledto=26042014
```

Settled payments after a date of more than a specific amount.

```
GET: /payments_settled?datesettledfrom=20042014&amountfrom=100
```

C# API Demo Project

Pay Advantage has produced a demonstration project to help you get started on your API integration. This project is written in C# and designed to run in Visual Studio. You will need .Net Framework 4.52 installed.

Start by downloading the project and extract the files into your working directory. Open the solution file in Visual Studio. Once opened, right-click on the solution and select ‘Enable NuGet Package Restore’. Rebuild the solution to download and restore required references. You may need to rebuild the solution once more after downloads have completed.

The demo application showcases the classes and methods contained within the “PayAdvantageAPI.dll” library. This file can be found within the “DLL” folder of the solution and referenced directly in your own applications if you prefer. The library provides HTTP and object level access to the API to simplify communication for you. It also has classes ready for you that represent the deserialised results of the API calls. For Windows Forms users a UI component is available using the same library. You will need to right-click on a Toolbox tab, click on “Choose Items” and browse to the DLL location.

The following properties can be set on the UI component to configure.

- PayAdvantageURL is the Url of the internet endpoint of the version of the API you are using.
- Username and Password are the API credentials that can be accessed from your Pay Advantage account. You will need administrator privileges to access this.
- WaitOnRequestOverLimit when set to true will cause the application to wait for the request limit period to renew in situations where you have exhausted the allowed number of requests per period. When set to false any such request over the limit would return an HTTP Status code of 403 (Forbidden) and error message “Request limit exceeded.”

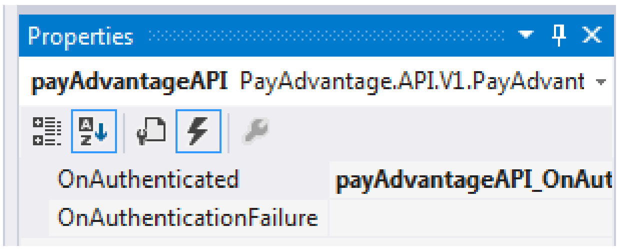
For those users implementing the API in other types of applications, you will need to provide the PayAdvantageURL, Username, and Password manually.

Authentication using the PayAdvantageAPI Library

The authentication with the API will happen automatically on the first request to the API as long as valid credentials have been set. The authentication token received will be maintained internally and attached to the header of subsequent requests. In a stateless environment such as an ASP.Net page cycle where the component would be created anew for every page cycle this would need to be persisted by some other means such as session and then the credentials and authentication token set. The authentication token is a string and can be set using a public method of the component

```
public void SetAuthToken(string token)
```

The component exposes two events to communicate the results of server authentication requests and also a Boolean Authenticated property.


Properties

payAdvantageAPI PayAdvantage.API.V1.PayAdvant

OnAuthenticated payAdvantageAPI_OnAut

OnAuthenticationFailure

Note:
These events are invoked from a background thread and therefore in a windows application it is necessary to use BeginInvoke() to switch to the main thread when updating windows components. See the demo application for an example of how to do this.

HTTP level access using the PayAdvantageAPI component

The PayAdvantageAPI component provides functions to assist in the calling of GET, POST, PUT and DELETE operations to the API. Provided the login credentials have been set on the component, authentication will be handled automatically as needed. If working in a stateless environment such as within the ASP.Net page cycle it will be necessary to take steps to persist the authentication token and restore the token using SetAuthToken() between page cycles to prevent the overhead of a fresh login to the API on each call.

```
public bool Authenticate(string username, string password)  auth token stored in AuthenticationToken property of component
public void SetAuthToken(string token)
public PaapiResponse Post(string action, string content)  action is the route to call and content is the request body contents
public PaapiResponse Put(string action, string content)
public PaapiResponse Put(string action)
public PaapiResponse Delete(string action)
public PaapiResponse Call(string action)          performs a GET operation
```

The methods above return a PaapiResponse object which contains the following public properties

```
public bool Succeeded { get; }                         True if the operation succeeded
public HttpResponseMessage Response { get; }           The HTTP response to the request
public string ResultContent { get; }                   JSON result object if call succeeded
public string ErrorMessage { get; set; }               Error message if the call failed
public int? X_Rate_Limit_Limit                         Current rate limit
public int? X_Rate_Limit_Remaining                     Request remaining in current period
public int? X_Rate_Limit_Reset                         Seconds until next rate period reset
```

PayAdvantage

API Object Layer

The PayAdvantageAPI component exposes classes for each of the object types that the API returns.

Customers

The customer class exposes the following methods for customer creation / retrieval

```
public PAAPIResult<Customer> Create(Customer newCustomer)
public PAAPIResult<Customer> Update(Customer updateCustomer)
public PAAPIResult<Customer> Get(string code)
public PAAPIResult<RecordList<Customer>> Get(string Query, int PageIndex, int PageSize)
```

The Get() method returning a RecordList takes a query parameter which is the same query string parameter as can be passed when accessing the API over HTTP. This can be used to reduce the size of the data request and streamline performance if only a subset of the data is required. The PageIndex and PageSize parameters work identically to the HTTP interface.

The results are returned as a generic class typed to the particular object being manipulated. The namespace for all objects is PayAdvantage.API.Models.

The PAAPIResult type returned as the result of the operation contains the following public properties.

`public bool Succeeded { get; }`	*True if the operation was successful*
`public T Result { get; }`	*The result object (could be a RecordList or the Typed object)*
`public string ErrorMessage { get; }`	*Contains the error message if the operation failed*
`public PaapiResponse Response { get; }`	*Contains the underlying HTTP call result*

Similar access methods exist for the other classes.

Payments

```
public PAAPIResult<Payment> Get(string code)
public PAAPIResult<RecordList<Payment>> GetSettled(int PageIndex, int PageSize, DateTime? date =
null)
public PAAPIResult<RecordList<Payment>> GetFailed(int PageIndex, int PageSize, DateTime? date =
null)
```

DebitBatches

```
public PAAPIResult<DebitBatch> Create(DebitBatch debitBatch)
public PAAPIResult<DebitBatch> Get(string code)
public PAAPIResult<RecordList<DebitBatch>> Get(string Query, int PageIndex, int PageSize)
public PAAPIResult<bool> Authorise(string code)
public PAAPIResult<bool> Confirm(string code)
```

DebitInstructions

```
public PaapiResponse Create(DebitInstructionList debitInstructionList)
public PAAPIResult<DebitInstruction> Get(string code)
public PAAPIResult<RecordList<DebitInstruction>> Get(string Query, int PageIndex, int PageSize)
public PAAPIResult<RecordList<DebitInstruction>> GetFailed(int PageIndex, int PageSize, DateTime?
date = null)
public PAAPIResult<bool> AckFail(string[] codes)        Acknowledge specific failed debit instructions
```

Field Description and Definitions

Please note that field names shown grey are NOT included in lists by default. You may include these fields by explicitly defining the fields to return or adding them to the "include" parameter. The order of the fields shown in this table DOES NOT guarantee the order of fields in results.

Customers

Name	*Description*	*Type*	*Min*	*Max*	*Sortable*	*Filterable*	*Editable*
Code	Unique internal identifier.	string	6	12	-	Y	-
ExternalID	External reference ID.	string	0	20	-	Y	Y
IsConsumer	Flag to indicate if this is a Consumer type customer or not. False means customer is a Business type.	bool			Y	Y	Y
DateJoined	Can be set to a date relevant to when the customer record entered your business.	date			Y	RANGE	Y
Name	Business Name or First + Last names of the customer. " Only editable for Business type customers.	string	1	80	Y	Y	Y*
FirstName	The first name of a consumer or first name of contact for a business	string	1	50	Y	Y	Y
LastName	The last name of a consumer or last name of contact for a business	string	1	50	Y	Y	Y
CustomRef	Customer reference.	string	0	20	Y	Y	Y
Email	Email address of the customer.	string	0	200	-	Y	Y
BillerCode	Biller Code of the customer.	string (ints only)	0	8	-	-	-
BPAYRef	BPAY Reference number for the customer.	string (ints only)	0	10	-	Y	-
DateUpdated	Date & time the customer record was updated.	datetime			Y	RANGE	-
DateCreated	Date & time the customer record was created.	datetime			Y	RANGE	-

CreatedBy.FirstName	First name of the account that created the customer.	string	2	50	Y	-	-
CreatedBy.LastName	Last name of the account that created the customer.	string	2	50	Y	-	-
CreatedBy.UserName	User name of the account that created the customer.	string	5	255	Y	-	-
IsActive	Flag to indicate an active customer	bool			-	Y	Y
URI	URI that directly accesses this customer record.	string		255	-	-	-

Debit Batches

Name	*Description*	*Type*	*Min*	*Max*	*Sortable*	*Filterable*	*Editable*
Code	Unique internal identifier.	string	6	12	-	Y	-
DateCreated	Date & time the debit batch was created.	datetime			Y	RANGE	-
Name	Name of the debit batch.	string	1	50	-	Y	Y
DateToDebit	Date to debit the debit batch.	date			Y	RANGE	Y
IsConfirmed	True / False the batch is confirmed.	bool			-	Y	-
IsAuthorised	True / False the batch is authorised.	bool			-	Y	-
IsProcessed	True / False the batch has been processed.	Bool			-	Y	-
RemitterName	Remitter Name of the batch.	string	0	16	-	-	Y
DebitInstructionCount	Number of debit instructions in the batch.	int			-	RANGE	-
DebitInstructionAmountSum	Total amount of the debit instructions in the batch.	decimal			-	RANGE	-
CreatedBy.FirstName	First name of the account that created the debit batch.	string	2	50	Y	Y	-
CreatedBy.LastName	Last name of the account that created the debit batch.	string	2	50	Y	Y	-
CreatedBy.UserName	User name of the account that created the debit batch.	string	5	255	Y	Y	-
URI	URI that directly accesses this debit batch record.	string		255	-	-	-

Debit Instructions

Name	*Description*	*Type*	*Min*	*Max*	*Sortable*	*Filterable*	*Editable*
Code	Unique internal identifier.	string	6	12	-	Y	-
Reference	Use this field as an optional placeholder for your own system ID	string	0	50	-	Y	Y
GroupKey	Use this field to group instructions together. Allows failed on-charged fees to be recovered on subsequent debit attempts.	string	0	50	-	Y	Y
DateCreated	Date & time the debit instruction was created.	datetime			Y	RANGE	-
IsProcessed	Flag to indicate if the debit instruction has been processed.	bool			-	Y	-
BSBNumber	BSB Number of the account to debit.	string (ints only)	6	6	-	Y	Y
AccountNumber	Account Number of the account to debit.	string (ints only)	5	9	-	Y	Y
AccountName	Account Name of the account to debit.	string	2	50	-	Y	Y
Amount	Amount to debit.	decimal(7,2)	5.0	99999.99	Y	Y	Y
RemitterName	Remitter of the debit instruction.	string	0	16	-	-	-
OnchargeFees	Flag to indicate if debit fees should be added to amount.	bool			-	Y	-
IsFailAcknowledged	Flag to indicate whether a failed debit instruction has been acknowledged.	bool			-	Y	-
DebitBatch.Code	Unique identifier of parent Debit batch.	string	6	12	-	Y	-
DebitBatch.DateToDebit	Date to Debit of the parent Debit batch.	date			Y	RANGE	-
Payment.Code	Unique internal identifier of the payment attached to this debit instruction.	string	6	12	-	Y	-
Payment.Amount	The amount of the actual payment (inclusive of fees)	decimal(7, 2)			-	Y	-
Payment.DatePaid	Payment date of the payment attached to this debit instruction.	date			Y	RANGE	-
Payment.DateFailed	Date of failure of the payment attached to this debit instruction.	date			Y	RANGE	-

URI	URI that directly accesses this debit instruction record.	string	0	255	-	-	-

Payments

Name	*Description*	*Type*	*Min*	*Max*	*Sortable*	*Filterable*	*Editable*
Code	Unique internal identifier.	string	6	12	-	Y	-
DatePaid	Date the payment was paid.	date			Y	RANGE	-
DateFailed	Date & time the payment failed.	datetime			Y	RANGE	-
FailCode	See failure codes.	char	1	1	-	Y	-
FailReason	Failure reason if the payment failed.	string	6	6	-	-	-
Amount	Payment amount inclusive of fees.	decimal(7,2)	5.0	99999.99	Y	RANGE	-
AmountExFees	Amount of the payment excluding any fees.	decimal(7,2)	5.0	99999.99	Y	RANGE	-
PaymentTypeID	1 for Direct Debit, 2 for BPAY, 8 For Debit instruction	byte	1	8	Y	Y	-
ExternalReference	The Reference applied to a related debit instruction	string	0	50	-	-	-
BPAYReference	If payment relates to a BPAY payment	string (ints only)	0	10	-	-	-
URI	URI that directly accesses this payment record.	string	0	255	-	-	-