domo-php Documentation

Wogan May

May 18, 2018

Table of Contents

CHAPTER 1

Installation

Install the package from composer:

```
$ composer require woganmay/domo-php
```

Once installed, you'll want to use the autoloader:

```
require "vendor/autoload.php";
```

The package is installed under the `WoganMay\DomoPHP` namespace.

CHAPTER 2

Authentication

The domo-php library needs a Client ID and Secret from the Domo Developer site. Log in at https://developer.domo.com/manage-clients to create a new client.

Once you have the ID and secret, you can use them to create a new instance of the API client:

```
use WoganMay\DomoPHP\DomoPHP;

$client_id     = "guid";
$client_secret = "sha256";

$client = new DomoPHP($client_id, $client_secret);
```

The new client should authorize all 4 scopes (data, audit, user and dashboard). If you want to use fewer scopes, you should indicate which you've created a client for as the third parameter, ie:

```
$client = new DomoPHP($client_id, $client_secret, ['data', 'user']);
```

CHAPTER 3

Overview

The domo-php library is organized into two parts - API and Helpers. Mostly you'll be using the API object to interact with the API itself. The Helpers streamline some of the more tedious tasks.

This distinction is made in the object structure itself:

```
$client = new DomoPHP($id, $secret);

$client->API; // Proxies for accessing the Domo API
$client->Helpers; // Helpers for the more tedious stuff
```

3.1 API

3.1.1 Activity Logs

The endpoint returns all the activity logs, sorted by event time descending. The basic call to fetch the last 50 events:

```
$entries = $client->API->Admin->getActivityLog();
```

To filter the results, you've got the following options:

```
$filters = [
    "user" => 12345,
    "start" => "2018-01-01 00:00:00",
    "end" => "2018-04-01 15:00:00",
    "limit" => 100,
    "offset" => 0
];

$filteredEntries = $client->API->Admin->getActivityLog($filters);
```

The service will automatically parse ISO timestamps into the epoch format the API needs. Or you can pass in epoch times yourself.

3.1.2 DataSets

This service lets you create, update and delete datasets. You can also import and export data as CSV, though it's advisable to limit this to smaller sets (<50MB). For really big loads you'll want to use the Streams API.

About the Schema

Every DataSet has a schema, which is defined as a list of fields of the following types:

- STRING
- LONG
- DOUBLE
- DECIMAL
- DATE
- DATETIME

When creating a schema, each field needs a unique, case-sensitive name, and the order of the schema fields are important when loading data. To make this a little easier, a fluent builder class is provided.

Create a new DataSet

A Domo DataSet consists of the metadata (title, schema, etc), and the records themselves. To create a new dataset, you first need to publish the schema, then follow that with a data import.

This sample uses the Schema Builder to set up a simple dataset:

```
$name = "My First Dataset";
$description = "An optional description";

$builder = $client->Helpers->SchemaBuilder->create();
$builder->date("Date");
$builder->string("Fruit");
$builder->double("Revenue");
$schema = $builder->toArray();

$dataset = $client->API->DataSet->createDataSet($name, $schema, $description);
```

A successful creation call will return the DataSet object from the API, and any failures will throw an Exception.

Fetch DataSet information

To fetch a list of DataSets, use the `getList()` method. The Limit and Offset can be used to page through all the datasets you have access to:

```
$limit = 50; // Maximum is 50 per call
$offset = 0;

$datasets = $client->API->DataSet->getList($limit, $offset);
```

There's currently no API method for filtering or searching for DataSets, so to find a particular set by name, you will need to pull all of them and page through.

Update DataSet Metadata

To change a DataSet's metadata (name, description or schema), use the `updateDataSet()` method:

```
$updates = [
  "name" => "New Name",
  "description" => "New Description"
];

$client->API->DataSet->updateDataSet($guid, $updates);
```

A successful update will return the DataSet object from the API, and any failures will throw an Exception.

Delete a DataSet

To irreversibly destroy a DataSet, use the `deleteDataSet()` method:

```
$client->API->DataSet->deleteDataSet($guid);
```

This will return a `TRUE` or `FALSE` depending on whether the DataSet was deleted.

Import data into a DataSet

To replace the data in a DataSet, use the `importDataSet()` method. This takes a headerless string variable of CSV content.

Important! The order of columns in the import data has to match the order of the dataset schema. Dates should be provided in ISO format (`yyyy-mm-dd`) to ensure there are no errors in parsing:

```
$csv = "2018-01-01,Apples,100.00\n2018-01-02,Apples,200.00";

$client->API->DataSet->importDataSet($guid, $csv);
```

There are limits to the amount of data you can load this way. If it's a few thousand records it'll be fine, but for larger loads (tens of thousands upwards), you'll want to use the Stream service.

Export data from a DataSet

You can use `exportDataSet()` to export the contents of a DataSet as CSV:

```
$exportHeaders = false; // Whether to include the header row (default: true)
$csv = $client->API->DataSet->exportDataSet($guid, $exportHeaders);
```

The resulting output can be written straight to a file on disk.

Working with PDP

domo-php includes a set of methods for working with PDP on a dataset:

- `getPDPList()`
- `getDataSetPDP()`
- `createDataSetPDP()`

- `updateDataSetPDP()`
- `deleteDataSetPDP()`

These will be documented in more detail at a later date. It doesn't look like the PDP system is accessible through the Domo UI anymore, so while the API is still creating policies, there's no way to interact with them through the UI anyway.

3.1.3 Groups

Groups are pretty simple - they're just containers that can hold users. There's the option to set a group as the "default" group for new users to join, but that method doesn't seem to work.

Creating and populating a group

Simple enough to create a group:

```
$name = "My Group";
$group = $client->API->Group->createGroup($name);
```

To populate the group, you will need the User IDs of the people you want to add. Users are added one at a time, by sending in the Group ID and the User ID to add:

```
$client->API->Group->addUser($group->id, 12345);
```

Users are removed from groups in a similar way:

```
$client->API->Group->removeUser($group->id, 12345);
```

Renaming a group

To rename a group, you just need its ID:

```
$client->API->Group->renameGroup($group->id, "New Name");
```

Activating and deactivating groups

If you need to activate or deactivate groups, there are simple methods for that:

```
$client->API->Group->activateGroup($group->id);
$client->API->Group->deactivateGroup($group->id);
```

Deleting a group

By deleting a group, it'll be removed from any pages or cards it's associated to. The users in the group won't be affected:

```
$client->API->Group->deleteGroup($group->id);
```

3.1.4 Pages

This service lets you work with pages and collections.

Getting existing pages

As with every other service, a `getList()` method lets you get a paginated list of existing pages:

```
$limit = 100; // Maximum: 500
$offset = 0;
$pages = $client->API->Page->getList($limit, $offset);
```

Creating Pages and Collections

Pages are a lot like Groups - containers for things. Creating them just requires a name:

```
$page = $client->API->Page->createPage("Page Name");
```

You can optionally pass in an array of properties. To nest a page, you'll want a parentId for another page.

To add a new collection to the system, you need the Page ID and the title:

```
$collection = $client->API->Page->createPageCollection($page->id, "My Collection");
```

Populating Pages and Collections

To assign cards to collections (or the pages they contain), you need to issue an update call with the IDs you want. There's a simple function for pages:

```
$client->API->Page->addCard($page->id, $card_id);
```

To do this for collections, you'll want to do an update:

```
$client->API->Page->updatePageCollection($page->id, [ 'cardIds' => [123,456] ]);
```

The same works for removing cards - just issue updates absent the card IDs you want to remove.

Deleting Pages and Collections

Deleting pages won't delete the cards themselves. Deleting a parent page won't cascade down to the child pages - they just become orphaned:

```
$client->API->Page->deletePage($page->id);
$client->API->Page->deletePageCollection($page->id, $collection->id);
```

3.1.5 Users

Getting Users

Use the `getList()` method to fetch existing users:

```
$limit = 10;
$offset = 0;
$users = $client->API->User->getList($limit, $offset);
```

Adding new users

When creating a new user, you need a primary email address (unique in your instance), and you have the option of sending an email invite or not.

You can't use this endpoint to set or change the user's password, so you'll usually want the invite sent. If you don't, the only way to set a password would be to go into the admin panel and use the Reset Password feature (or have the user do a self-service reset).

To create a user with all the defaults:

```
$name = "John Doe";
$email = "john.doe@example.org";
$user = $client->API->User->createUser($name, $email);
```

That will create a Participant user with no additional attributes, without sending an invite. To do a full-on onboarding:

```
$profile = [
    "title" => "Junior Something",
    "mobile" => "+18001234567",
    "employeeNumber" => "007"
];
$sendInvite = true;
$user = $client->API->User->createUser("Full User", "fulluser@example.org",
↪"Privileged", $profile, $sendInvite);
```

This creates a user with some prepopulated profile fields, and dispatches an email invite.

Updating Users

Important! There's an oddity with this endpoint. In order to do an incremental update, you need to specify the user's existing email addresses:

```
$client->API->User->updateUser(123, "john.doe@example.org", [ "title" => "Senior␣
↪Something" ]);
```

Deleting Users

To delete a user, you just need the ID:

```
$client->API->User->deleteUser(123);
```

3.1.6 Streams

Streams allow you to upload large amounts of data in split, compressed, sequential files. Each upload is handled as a single 'execution' - a stream is started, populated, then committed. In the background, Domo will aggregate all the parts you uploaded and produce a final dataset.

This is the recommended approach for loading really large sets of data. The streams can be created in REPLACE or APPEND mode, depending on your ingestion strategy.

By default, Domo Workbench will load 100‘000 rows per upload. It’s probably best to follow that guideline with the Streams API.

Listing Streams

To get a listing of all the Streams already created:

```
$limit = 10;
$offset = 0;
$client->API->Streams->getList($limit, $offset);
```

This will return a list (limit and offset parameters are optional).

Getting Streams

To get a single Stream, fetch it by numeric ID:

```
$stream_id = 123;
$stream = $client->API->Streams->getStream($stream_id);
```

This will return a Stream object, which is composed of the basic Stream metadata, and a description of the DataSet produced by the Stream.

Create a new Stream

Creating a new Stream is a lot like creating a new DataSet - it requires the same information as a DataSet, in addition to an update mode:

```
$builder = $client->Helpers->SchemaBuilder->create();
$builder->string("Full Name");
$builder->double("Salary");
$builder->date("Start Date");
$columns = $builder->toArray();

$stream = $client->API->Stream->createStream("Test Dataset", $columns);
```

This will return the Stream object. Using the ID, you can now start populating the Stream.

Update a Stream

The only update you can make to a Stream is to change the update mode (REPLACE or APPEND):

```
$stream_id = 123;
$stream = $client->API->Stream->updateStream($stream_id, "APPEND");
```

On a successful update, the client will return the Stream object.

Delete a Stream

Deleting a Stream is straightforward, and will delete the associated DataSet:

```
$stream_id = 123;
$result = $client->API->Stream->deleteStream($stream_id);
```

The client will return TRUE on a successful delete.

List Stream Executions

Every time you need to send new data into a Stream, it's managed as an Execution. Executions are created, populated, and committed, or aborted if something went wrong.

To list the existing executions for a single stream:

```
$stream_id = 123;
$executions = $client->API->Stream->listStreamExecutions($stream_id);
```

Getting Stream Executions

To get a single Stream execution:

```
$stream_id = 123;
$execution_id = 1;

$execution = $client->API->Stream->getStreamExecution($stream_id, $execution_id);
```

This will return a single Stream execution.

Create a new Stream Execution

If you create a new Execution before committing or aborting an existing Execution, your new execution will obliterate the old one. It's recommended to abort an execution you intend on abandoning.

To create a new Execution:

```
$stream_id = 123;
$execution = $client->API->Stream->createStreamExecution($stream_id);
```

This returns a new Execution, which will be automatically set to the update mode of the underlying Stream (REPLACE or APPEND). To change the update mode, you'll need to abort this execution and update the Stream first.

Upload data to an Execution

The Stream already has your data schema (at the point where the Stream was created), so you just need to provide CSV files with the columns in a matching order.

Every file has a Part number (starts at 1) that indicates its position in the Stream. You don't need to declare the total number of parts upfront, but it's important to upload an unbroken sequence:

```
// Upload 3 data parts
$stream_id = 123;
$execution_id = 1;

$part1 = file_get_contents("part1.csv");
$part2 = file_get_contents("part2.csv");
$part3 = file_get_contents("part3.csv");

$part1_result = $client->API->Stream->uploadData($stream_id, $execution_id, 1,
↪$part1);
$part2_result = $client->API->Stream->uploadData($stream_id, $execution_id, 2,
↪$part1);
$part3_result = $client->API->Stream->uploadData($stream_id, $execution_id, 3,
↪$part1);
```

Once all the Upload calls are finished, the Stream is ready to be committed.

You can upload the sequence in parallel - so for tools that support it, several upload calls can run at once. Parts can also be re-tried if a single part upload fails.

Commit a Stream Execution

Once all your parts are uploaded, commit the Stream Execution to process them:

```
$stream_id = 123;
$execution_id = 1;

$result = $client->API->Stream->commitStreamExecution($stream_id, $execution_id);
```

Depending on the amount of data uploaded, this could take a few minutes to process. Once the Stream is processed, the resulting Dataset will become available in your Domo Data Center.

Abort a Stream Execution

If you need to abort a Stream Execution:

```
$stream_id = 123;
$execution_id = 1;

$result = $client->API->Stream->abortStreamExecution($stream_id, $execution_id);
```

This will cancel all processing and throw away the data parts.

3.2 Helpers

The following helpers are available:

3.2.1 DataSet Importer

If you have a CSV file on disk that you want to quickly load into Domo, there's a straightforward helper method for that:

```
$file = "/path/to/import.csv";
$name = "Dataset Name";

$dataSet = $client->Helpers->DataSet->createDataSet($name, $file);
```

The resulting `$dataSet` will be the API object, or the method will throw an exception.

3.2.2 Schema Builder

Build a new Schema

To create a new schema array, you can use an instance of the SchemaBuilder helper. Chain together all the fields you need, then export it to an array:

```
$builder = $client->Helpers->SchemaBuilder->create();

$builder->string("Full Name");
$builder->date("Start Date");
$builder->long("Mobile Number");
$builder->decimal("Monthly Salary");
$builder->double("Hourly Rate");
$builder->datetime("Last Login At");
$schema = $builder->toArray();
```

The `$schema` array will be created in the same order that the builder methods are called.

Create a Schema from sample data

If you have the first two rows of a CSV file, there's a helper that can take a guess at the schema. You'll need to parse them from the import file yourself:

```
$headers = [ "Date",       "Fruit",  "Revenue" ];
$record  = [ "2018-01-01", "Apples", 100.00    ];

$schema = $client->Helpers->SchemaBuilder->inferSchema($headers, $record);
```

The resulting `$schema` can be used in a `createDataSet()` call.

If you run into any trouble with this library, you can ask for help in the chat (it's free to join!): https://spectrum.chat/domo-php