Webservice one-page documentation

Table of contents

Webservice one-page documentation

This chapter aims at grouping all the information from the webservice tutorial into a handy one-page doc that you can print and keep at hand.

Setting everything up

In order for you to view, edit and delete the data on your PrestaShop store through its webservice, you first need to enable the webservice feature, and then create an access key.

Enabling cURL

Configure your PHP installation so that it has the cURL extension installed and enabled:

With Windows, place this line in your `php.ini` file:

```
extension=php_curl.dll
```

With Linux/Mac, install the cURL extension:

```
sudo apt-get install php5-curl
```

Enabling the webservice feature

Go in the PrestaShop back-office, open the "Webservice" page under the "Advanced Parameters" menu, and then choose "Yes" for the "Enable PrestaShop's webservice". Save your change: you're done!

Creating an access key

Open the "Webservice" page under the "Advanced Parameters" menu, and then click the "Add New" button to access the account configuration section. A long form appears:

- **Key**. The API key serves as the main identifier for the webservice account you are creating. Click the "Generate" button to get an unique authentication key. You can also create your own (which must be 32 characters long), but using a generated key prevents wrong-doers from guessing your key too easily.
  Using this key, you and other selected users will be able to access the webservice.

- **Key description**. Helps you remember who you created that key for, what are the access rights assigned to it, etc. The description is not public, but make sure to put all the keywords pertaining to the user, so that you can find their key more quickly.
- **Status**. You can disable any key at any time.
- **Permissions**. This section is very important, as it enables you to assign rights for each resource you want to make available to this key. Indeed, you might want a user to have read and write access on some resources, but only read access on others – and no access to the more important ones.
  In the list of permissions, the checkbox most on the left enables you to define all the rights for a given resource. Likewise, the checkbox at the top of each column enables you to give the select right (View, Modify, etc.) to all the resources.
  Make sure to only select the rights needed for the usage of that key. Do not give all the rights for all resources to any key, keep that to yours and yours only.
- **Shop association**. This only appears in multistore mode. It enables you to choose which of your stores the key owner should have access to.

If you choose to use a custom passkey instead of a generated one, make sure it is very secure and that its rights are limited – and that it is 32characters long!

Accessing the webservice from the browser

The endpoint to your store's webservice is located in the `/api/` folder at the root of your installation of Prestashop:

- If PrestaShop is installed at the root of your server, you can access the API here: http://example.com/api/
- If PrestaShop is installed in a subfolder of your server, you can access the API here: http://example.com/prestashop/api/

To access it, you need to provide your API key when request. There is no password, providing your API key is enough – and therefore the key should be kept secret by the user!

You can either type the API endpoint address directly then enter your API key, or indicate your API key in the address. Here is an example, with "UCCLLQ9N2ARSHWCXLT74KUKSSK34BFKX" being the API key.

- At the root of the server: http://UCCLLQ9N2ARSHWCXLT74KUKSSK34BFKX@example.com/api/
- In a subfolder of the server: http://UCCLLQ9N2ARSHWCXLT74KUKSSK34BFKX@example.com/prestasshop/api/

You can test this with any browser that supports XML.

If no permission has been set for the key, then the browser will keep asking you to enter the key indefinitely.

If you cannot access some resources, the API might answer with this XML response:

```
<?xml version="1.0" encoding="UTF-8"?>
<prestashop xmlns:xlink="http://www.w3.org/1999/xlink">
  <errors>
    <error>
      <message><![CDATA[Internal error. To see this error please display the PHP errors.]]></message>
    </error>
  </errors>
</prestashop>
```

Using XLink, you will then be able to access your various resources. XLink associates an XML file to another XML file via a link.

Accessing the webservice use our PHP library

You will need the latest version of the `PSWebServiceLibrary.php` file, which can be found on our code repository: https://github.com/PrestaShop/PrestaShop-webservice-lib/blob/master/PSWebServiceLibrary.php
To download the file:

- Click here to view the raw file: https://raw.github.com/PrestaShop/PrestaShop-webservice-lib/master/PSWebServiceLibrary.php
- Copy/paste the file into an empty text local file, using for instance Notepad.

- Save the file as `PSWebServiceLibrary.php`.

You can also directly download a zip archive of all the files in this repository, including the example files, by clicking here: https://github.com/PrestaShop/PrestaShop-webservice-lib/archive/master.zip

Put `PSWebServiceLibrary.php` at the root of your web server, and add this line at the beginning of any PHP file that needs to use your store's webservice.

Then, you must create an instance of the `PrestaShopWebservice` object, which takes 3 parameters in its constructor:

- The store's root path (ex: http://example.com/).
- The authentication key (ex: UCCLLQ9N2ARSHWCXLT74KUKSSK34BFKX).
- A boolean value, indicating whether the webservice must use its debug mode.

Here's how you create a webservice call:

```
$webService = new PrestaShopWebservice('http://example.com/', 'UCCLLQ9N2ARSHWCXLT74KUKSSK34BFKX', false);
```

Handling errors

It is essential that you understand how to handle errors with the webservice library. By implementing error-catch method early, you will more easily detect issues, and be able to correct them on the go.

Error handling with the webservice library is done using PHP exceptions. If you do not know about them, you should read http://php.net/manual/en/language.exceptions.php, as exceptions are an essential part of good coding practice.

How it works

The error handling is done within a `try..catch` block, with the webservice processing being done in the try section, the catch one containing the error handling code.

```
try {
  // Execution ( if an error occurs in this code, stops and goes in the catch block)
}
catch {
  // Error handling (tries to catch the error or the error display)
}
```

Example

That means each creation or use of the library must be located within a "try" block. The "catch" block can then handle the error if it occurs during the execution of the try block.
Now we'll see how to list all customers via the webservice, and then we will see the four CRUD methods.

In the following code sample, we want to get the list of all customers:

- In the try block, we instantiate the `PrestaShopWebservice` object with the necessary parameters and retrieve the customer resource XML in a variable.
- In the catch block, we put code to display the PHP error message, if anything wrong happens in the try block.

```
try {
   // creating webservice access
   $webService = new PrestaShopWebservice('http://example.com/', 'UCCLLQ9N2ARSHWCXLT74KUKSSK34BFKX', false);

   // call to retrieve all clients
```

```
   $xml = $webService->get(array('resource' => 'customers'));
}
catch (PrestaShopWebserviceException $ex) {
   // Shows a message related to the error
   echo 'Other error: <br />' . $ex->getMessage();
}
```

Examples files

All the example files can be found on our code repository: https://github.com/PrestaShop/PrestaShop-webservice-lib/tree/master/examples

Working with your data

As shown above, you have to instantiate the `PrestaShopWebservice` object in order to use its methods: `add()`, `get()`, `edit()` and `delete()`.
You can also directly use call the REST API and get XML results, as this correspondence table shows:

CRUD operations	SQL statements	PrestaShopWebservice methods	HTTP methods
Create	INSERT	add()	POST
Read	SELECT	get()	GET
Update	UPDATE	edit()	PUT
Delete	DELETE	delete()	DELETE

Viewing your data

Using the PrestaShopWebservice methods

Let's see how to view a full list of client IDs. As you can see from the table above, we need the `get()` method in order to retrieve a XML file containing all the customers. The parameter has to be a key-value array, where we define the resource we want:

Key	Value
resource	customers

The value defines the resource that the webservice will use in a future call. The value could be carrier types, countries or any other type of resource that can be found in the "Available resources" section of this guide.

The code

In order to retrieve a list of customers in XML format, you only need to instantiate `PrestaShopWebservice` (a seen earlier) and use this code:

```
// The key-value array
$opt['resource'] = 'customers';

Retrieving the XML data
$xml = $webService->get($opt);
```

The result

Launching the code above will return a XML object containing all the current customer IDs on the store:

```
<?xml>
<prestashop>
  <customer id="1" xlink:href="http://example.com/api/customers/1"/>
  <customer id="2" xlink:href="http://example.com/api/customers/2"/>
  <customer id="3" xlink:href="http://example.com/api/customers/3"/>
  <customer id="4" xlink:href="http://example.com/api/customers/4"/>
</prestashop>
```

With this information, you can access the customer that interests you by using the value in the `id` attribute:

```
$opt['resource'] = 'customers';
$opt['id'] = 1;
$xml = $webService->get($opt);
```

Structure

As you can see in the result in the previous section, the data returned by calling `$webService->get()` puts you at the root of the XML document, in the context you requested.

To access the fields of clients who are children of the `<customers>` tag, we only need to retrieve all fields in an associative array in XML, like this:

```
$resources = $xml->customers->children();
```

From there, you can easily access client IDs. Here's an example with a path from identifiers:

```
foreach ($resources as $resource)
    echo $resource->attributes() . '<br />';
```

Thanks to these elements, you can create a HTML table containing all the client IDs – or anything that displays this information in your interface.

Using HTTP requests

If you would rather not use the `PrestaShopWebservice` object, note that PrestaShop's webservice is RESTful, in that you can work with its data using the known HTTP verbs just as easily as you would use the methods of the `PrestaShopWebservice` object.

Viewing your data follows the same rule, but with a HTTP GET request on the same URL:
To view your data you simply have to launch an HTTP GET request on the following URL:

```
http://UCCLLQ9N2ARSHWCXLT74KUKSSK34BFKX@example.com/api/customers/
```

The resulting XML document is the same as the one obtained using `PrestaShopWebservice`'s `get()` method.

Creating and adding

Creating and adding data takes a bit more work. Indeed, in order to create a new entry or edit an existing one, you need to send a fully formed and complete XML document, whether you use `PrestaShopWebservice` or the HTTP methods.

When you need to create a new entry, the API can help you by providing a blank schema for any resource, or even a synopsis with indications of the meaning for each tag:

```
// Returns a blank XML document, with all the tags ready to fill
http://UCCLLQ9N2ARSHWCXLT74KUKSSK34BFKX@example.com/api/manufacturers?schema=blank

// Returns a blank XML document, with all the tags ready to fill and indication of the expected value for each
http://UCCLLQ9N2ARSHWCXLT74KUKSSK34BFKX@example.com/api/manufacturers?schema=synopsis
```

⚠ When a client is created from within PrestaShop's administration interface, a confirmation e-mail is sent to the client. This cannot be done directly with the webservice: there is no way to trigger the sending of that confirmation e-mail.

However, you can create an override file for the `Customer` class and override the `addWs()` method. This method is similar to `ObjectModel::add()` but is only called from the webservice. You can find examples of its use in the `Product` and `Order` classes.

Deleting

The RESTful-ness of the API goes all the way: in order to delete the product with an ID of 12, you simply have to launch an HTTP DELETE request on the following URL:

```
http://UCCLLQ9N2ARSHWCXLT74KUKSSK34BFKX@example.com/api/products/12/
```

Available resources

The `/api/` URL gives you the root of all the resources, in the form of an XML file.

Here it is, extremely simplified. This list is current as of version 1.5.4.1 of PrestaShop. Note that we only show the API resources available for one of the available stores.

```
<?xml version="1.0" encoding="UTF-8"?>
<prestashop xmlns:xlink="http://www.w3.org/1999/xlink">
  <api shop_name="MYSHOP">
    <addresses>...</addresses>
    <carriers>...</carriers>
    <cart_rules>...</cart_rules>
    <carts>...</carts>
    <categories>...</categories>
    <combinations>...</combinations>
    <configurations>...</configurations>
    <contacts>...</contacts>
    <content_management_system>...</content_management_system>
    <countries>...</countries>
    <currencies>...</currencies>
    <customer_messages>...</customer_messages>
    <customer_threads>...</customer_threads>
    <customers>...</customers>
    <deliveries>...</deliveries>
    <employees>...</employees>
    <groups>...</groups>
    <guests>...</guests>
```

```
    <image_types>...</image_types>
    <images>...</images>
    <languages>...</languages>
    <manufacturers>...</manufacturers>
    <order_carriers>...</order_carriers>
    <order_details>...</order_details>
    <order_discounts>...</order_discounts>
    <order_histories>...</order_histories>
    <order_invoices>...</order_invoices>
    <order_payments>...</order_payments>
    <order_states>...</order_states>
    <orders>...</orders>
    <price_ranges>...</price_ranges>
    <product_feature_values>...</product_feature_values>
    <product_features>...</product_features>
    <product_option_values>...</product_option_values>
    <product_options>...</product_options>
    <product_suppliers>...</product_suppliers>
    <products>...</products>
    <search >...</search>
    <shop_groups>...</shop_groups>
    <shops>...</shops>
    <specific_price_rules>...</specific_price_rules>
    <specific_prices>...</specific_prices>
    <states>...</states>
    <stock_availables>...</stock_availables>
    <stock_movement_reasons>...</stock_movement_reasons>
    <stock_movements>...</stock_movements>
    <stocks>...</stocks>
    <stores>...</stores>
    <suppliers>...</suppliers>
    <supply_order_details>...</supply_order_details>
    <supply_order_histories>...</supply_order_histories>
    <supply_order_receipt_histories>...</supply_order_receipt_histories>
    <supply_order_states>...</supply_order_states>
    <supply_orders>...</supply_orders>
    <tags>...</tags>
    <tax_rule_groups>...</tax_rule_groups>
    <tax_rules>...</tax_rules>
    <taxes>...</taxes>
    <translated_configurations>...</translated_configurations>
    <warehouse_product_locations>...</warehouse_product_locations>
    <warehouses>...</warehouses>
    <weight_ranges>...</weight_ranges>
    <zones>...</zones>
  </api>
  <api shop_name="MYOTHERSHOP">...</api>
  <api shop_name="YETANOTHERSHOP">...</api>
</prestashop>
```

As with any XLink-bearing XML file, each resource element leads to more resources, linked from the `xlink` attribute.

```
<customers xlink:href="http://example.com/api/customers" get="true" put="true" post="true" delete="true" head="
true">
  <description xlink:href="http://example.com/api/customers" get="true" put="true" post="true" delete="true"
head="true">The e-shop's customers</description>
  <schema xlink:href="http://example.com/api/customers?schema=blank" type="blank"/>
  <schema xlink:href="http://example.com/api/customers?schema=synopsis" type="synopsis"/>
</customers>
```

In addition, the element contains a description of the resource, and two schemas: a blank one, which you can use to create a new item, and a synopsis one, which is just like the blank one but with a detailed description of what type of data is expected in each element.

Here is an extract of the Customer blank schema:

Blank schema

```
<?xml version="1.0" encoding="UTF-8"?>
<prestashop xmlns:xlink="http://www.w3.org/1999/xlink">
  <customer>
    <id></id>
    <id_default_group></id_default_group>
    <id_lang></id_lang>
    <newsletter_date_add></newsletter_date_add>
    <ip_registration_newsletter></ip_registration_newsletter>
    <last_passwd_gen></last_passwd_gen>
    <secure_key></secure_key>
    <deleted></deleted>
    <passwd></passwd>
    <lastname></lastname>
    <firstname></firstname>
    <email></email>
    ...
  </customer>
</prestashop>
```

Here is an extract of the Customer synopsis schema:

Synopsis schema

```
<?xml version="1.0" encoding="UTF-8"?>
<prestashop xmlns:xlink="http://www.w3.org/1999/xlink">
  <customer>
    <id_default_group></id_default_group>
    <id_lang format="isUnsignedId"></id_lang>
    <newsletter_date_add></newsletter_date_add>
    <ip_registration_newsletter></ip_registration_newsletter>
    <last_passwd_gen></last_passwd_gen>
    <secure_key format="isMd5"></secure_key>
    <deleted format="isBool"></deleted>
    <passwd required="true" maxSize="32" format="isPasswd"></passwd>
    <lastname required="true" maxSize="32" format="isName"></lastname>
    <firstname required="true" maxSize="32" format="isName"></firstname>
    <email required="true" maxSize="128" format="isEmail"></email>
    ...
  </customer>
</prestashop>
```

The value types can be found in the webservice reference: http://doc.prestashop.com/display/PS15/Web+service+reference.