Getting Started with the Force.com Toolkit for PHP

Abstract

PHP is one of the most widely used server-side programming languages currently in use, powering the user-facing portions of major websites including Facebook, Wikipedia and WordPress, and, alongside Perl and Python, the 'P' in the LAMP stack. This article covers the Force.com Toolkit for PHP for version 20.0 of the SOAP API, focusing on getting you off to a quick start accessing Force.com data from your PHP applications.

Introduction

The Force.com PHP Toolkit provides an easy-to-use wrapper for the Force.com Web Services SOAP API, presenting SOAP client implementations for both the enterprise and partner WSDLs. Since the enterprise WSDL is strongly typed, the enterprise client provides a very succinct syntax for manipulating records. In contrast, the partner WSDL provides a more generic interface, giving the ability to access data of any schema, but at the expense of a little more code. See Choosing a WSDL in the SOAP API Developer's Guide for a more detailed discussion of the two WSDL options.

Note that the *enterprise* WSDL included in the PHP toolkit references only the default Salesforce.com objects - follow these instructions to generate an enterprise WSDL specific to your organization. Since the *partner* WSDL is independent of the objects in use, you can just go ahead and use the version in the toolkit.

Regardless of whether you choose the enterprise or partner WSDL and associated SOAP client, this article will show you how to quickly get started manipulating data in Force.com from your PHP application.

Requirements

The Force.com PHP Toolkit requires PHP 5.x with the cURL, SOAP and OpenSSL PHP modules. The toolkit has been tested with a wide variety of PHP 5.x versions on a range of operating systems. If you encounter problems that appear to be related to your particular PHP version or OS, let us know on the Perl, PHP, Python & Ruby Development Force.com Discussion Board.

Installing the Toolkit

Clone the PHP Toolkit code from Github with

```
git clone https://github.com/developerforce/Force.com-Toolkit-for-PHP.git
```

If the command above does not work, download it from https://github.com/developerforce/Force.com-Toolkit-for-PHP instead.

Copy the soapclient directory to your PHP application's directory - for a web application this will typically be in the web server document tree, for example `/var/www/html/myphpapp`.

Creating a Connection

The first step to using the PHP Toolkit is to create a connection using either `SforceEnterpriseClient` or `SforcePartnerClient`. In both cases, you must ensure that your user has API access enabled (Setup|Manage Users|Users|Select your user|Click the user's profile|Administrative Permissions), and concatenate the password with the security token to form the second argument for the `login` method. Note – in this, and the other code fragments presented in this guide, error handling is omitted for clarity. The sample code includes basic error handling.

Enterprise

```
define("USERNAME", "user@example.com");
define("PASSWORD", "password");
define("SECURITY_TOKEN", "sdfhkjwrhgfwrgergp");

require_once ('soapclient/SforceEnterpriseClient.php');

$mySforceConnection = new SforceEnterpriseClient();
$mySforceConnection->createConnection("enterprise.wsdl.xml");
$mySforceConnection->login(USERNAME, PASSWORD.SECURITY_TOKEN);
```

Partner

```
define("USERNAME", "user@example.com");
define("PASSWORD", "password");
define("SECURITY_TOKEN", "sdfhkjwrhgfwrgergp");

require_once ('soapclient/SforcePartnerClient.php');

$mySforceConnection = new SforcePartnerClient();
$mySforceConnection->createConnection("partner.wsdl.xml");
$mySforceConnection->login(USERNAME, PASSWORD.SECURITY_TOKEN);
```

Executing a Query

Executing a SOQL query is straightforward with either client; the only difference is that, with the partner client, the fields must be accessed via an intermediate `fields` object:

Enterprise

```php
$query = "SELECT Id, FirstName, LastName, Phone from Contact";
$response = $mySforceConnection->query($query);

echo "Results of query '$query'<br/><br/>\n";
foreach ($response->records as $record) {
    echo $record->Id . ": " . $record->FirstName . " "
        . $record->LastName . " " . $record->Phone . "<br/>\n";
}
```

Partner

```php
$query = "SELECT Id, FirstName, LastName, Phone from Contact";
$response = $mySforceConnection->query($query);

echo "Results of query '$query'<br/><br/>\n";
foreach ($response->records as $record) {
    // Id is on the $record, but other fields are accessed via the fields object
    echo $record->Id . ": " . $record->fields->FirstName . " "
        . $record->fields->LastName . " " . $record->fields->Phone . "<br/>\n";
}
```

Creating Records

The enterprise client's `create` method accepts an array of generic PHP objects and their type, where PHP object properties correspond to record fields. The `create` method returns an array containing the result for each record creation:

Enterprise

```php
$records = array();

$records[0] = new stdclass();

$records[0]->FirstName = 'John';
$records[0]->LastName = 'Smith';
$records[0]->Phone = '(510) 555-5555';
$records[0]->BirthDate = '1957-01-25';

$records[1] = new stdclass();
$records[1]->FirstName = 'Mary';
$records[1]->LastName = 'Jones';
$records[1]->Phone = '(510) 486-9969';
$records[1]->BirthDate = '1977-01-25';

$response = $mySforceConnection->create($records, 'Contact');

$ids = array();
foreach ($response as $i => $result) {
    echo $records[$i]->FirstName . " " . $records[$i]->LastName . " "
            . $records[$i]->Phone . " created with id " . $result->id
            . "<br/>\n";
    array_push($ids, $result->id);
}
```

In contrast, the partner client's `create` method takes an array of `SObjects`, where each `SObject` contains an array of fields and has its type set individually:

Partner

```php
$records = array();

$records[0] = new SObject();
$records[0]->fields = array(
    'FirstName' => 'John',
    'LastName' => 'Smith',
    'Phone' => '(510) 555-5555',
    'BirthDate' => '1957-01-25'
);
$records[0]->type = 'Contact';

$records[1] = new SObject();
$records[1]->fields = array(
    'FirstName' => 'Mary',
    'LastName' => 'Jones',
    'Phone' => '(510) 486-9969',
    'BirthDate' => '1977-01-25'
);
$records[1]->type = 'Contact';

$response = $mySforceConnection->create($records);

$ids = array();
foreach ($response as $i => $result) {
    echo $records[$i]->fields["FirstName"] . " "
            . $records[$i]->fields["LastName"] . " "
            . $records[$i]->fields["Phone"] . " created with id "
            . $result->id . "<br/>\n";
    array_push($ids, $result->id);
}
```

Retrieving Records

The `retrieve` method works similarly for the enterprise and partner clients, taking a list of required fields, the record type, and an array of record id's. The only difference between the clients is that records returned by the partner client have a `fields` object containing the fields themselves:

Enterprise

```php
// $ids is an array of record ids built in the previous step
```

```
$response = $mySforceConnection->retrieve('Id, FirstName, LastName, Phone',
                'Contact', $ids);

foreach ($response as $record) {
    echo $record->Id . ": " . $record->FirstName . " "

    . $record->LastName . " " . $record->Phone . "<br/>\n";
}
```

Partner

```
// $ids is an array of record ids built in the previous step
$response = $mySforceConnection->retrieve('Id, FirstName, LastName, Phone',
                'Contact', $ids);
foreach ($response as $record) {
    echo $record->Id . ": " . $record->fields->FirstName . " "
    . $record->fields->LastName . " " . $record->fields->Phone . "<br/>\n";
}
```

Updating Records

The `update` method is very similar to the `create` method for both clients; simply supply the fields to be updated:

Enterprise

```
$records[0] = new stdclass();
$records[0]->Id = $ids[0];
$records[0]->Phone = '(415) 555-5555';

$records[1] = new stdclass();
$records[1]->Id = $ids[1];
$records[1]->Phone = '(415) 486-9969';

$response = $mySforceConnection->update($records, 'Contact');
foreach ($response as $result) {
    echo $result->id . " updated<br/>\n";
}
```

Partner

```
$records[0] = new SObject();
$records[0]->Id = $ids[0];
$records[0]->fields = array(
    'Phone' => '(415) 555-5555',
);
$records[0]->type = 'Contact';

$records[1] = new SObject();
$records[1]->Id = $ids[0];
$records[1]->fields = array(
    'Phone' => '(415) 486-9969',
);
$records[1]->type = 'Contact';

$response = $mySforceConnection->update($records);
foreach ($response as $result) {
    echo $result->id . " updated<br/>\n";
}
```

Note that, due to the way that serialization works, setting a field to null will leave the field in place as an empty string (""). To actually *remove* a field from a record, setting it to null in the database, use the `fieldsToNull` property as follows:

Enterprise

```
$records[0] = new stdclass();
$records[0]->Id = $ids[0];
$records[0]->fieldsToNull = 'Phone';

$records[1] = new stdclass();
$records[1]->Id = $ids[1];
$records[1]->fieldsToNull = 'Phone';


$response = $mySforceConnection->update($records, 'Contact');
```

Partner

```
$records[0] = new SObject();
$records[0]->Id = $ids[0];
$records[0]->fieldsToNull = 'Phone';
$records[0]->type = 'Contact';

$records[1] = new SObject();
$records[1]->Id = $ids[1];
$records[1]->fieldsToNull = 'Phone';
$records[1]->type = 'Contact';

$response = $mySforceConnection->update($records);
```

If you wish to remove multiple fields, pass an array in the `fieldsToNull` property:

```
$records[0]->fieldsToNull = array('Phone', 'Fax');
```

Deleting Records

The `delete` method takes an array of record id's; the enterprise and partner clients implement it identically:

Enterprise/Partner

```
// $ids is an array of record ids built in a previous step
$response = $mySforceConnection->delete($ids);
foreach ($response as $result) {
    echo $result->id . " deleted<br/>\n";
}
```

Session Management

When the user navigates from one PHP page to another, the SOAP client must be re-instantiated for each page. In the first PHP page after a successful login, store the Session ID, endpoint, and WSDL reference in the session:

```
// session_start() usually goes at the beginning of the PHP script
session_start();

// [Code to create the connection...]

// Now we can save the connection info for the next page
$_SESSION['location'] = $mySforceConnection->getLocation();
$_SESSION['sessionId'] = $mySforceConnection->getSessionId();
$_SESSION['wsdl'] = $wsdl;
```

On subsequent PHP pages, re-instantiate the SOAP client by retrieving the attributes that were stored in the session:

```
session_start();

// Retrieve session attributes
$location = $_SESSION['location'];
$sessionId = $_SESSION['sessionId'];
$wsdl = $_SESSION['wsdl'];

// Use SforceEnterpriseClient or SforcePartnerClient as appropriate
$mySforceConnection = new SforceEnterpriseClient();
$mySforceConnection->createConnection($wsdl);
$mySforceConnection->setEndpoint($location);
$mySforceConnection->setSessionHeader($sessionId);

// Now the connection is ready for use...
```

Summary

This article covered the basics of manipulating records with the PHP toolkit: query, create, retrieve, update and delete. For more details, see the API documentation included in the toolkit `apidocs` directory.

Related Links

- Force.com Toolkit for PHP on Developer Force
- tksample.php Sample PHP application showing the features described in this article
- PHP Toolkit 20.0 Samples provides sample code for most of the API calls
- Force.com Toolkit for PHP Github Project
- Web Services API Developer's Guide
- Perl, PHP, Python & Ruby Development Force.com Discussion Board

About the Author

Pat Patterson is a recent addition to the Developer Evangelism team at Salesforce.com, currently focusing on the Force.com APIs. Describing himself as an 'articulate techie', Pat hacks all manner of code from Ruby web apps down to Linux kernel drivers, writing it all up on the Force.com blog, his own blog Superpatterns, and, of course, Twitter.