Getting Started with the Bullhorn SOAP API and PHP

Introduction

This tutorial is for developers who develop custom applications that use Bullhorn SOAP API and PHP. You develop a sample application that creates a session to the Bullhorn system, retrieves candidates by using several methods, and displays the results in a table on the web page.
You learn how to do the following tasks:

- Set up your environment for development.
- Access the Bullhorn SOAP API.
- Create and publish a web application in Eclipse that retrieves candidate data.

You can follow the step-by-step instructions and/or download and run the provided code sample files.

Prerequisites

Bullhorn recommends the following prerequisites for this tutorial:

- Knowledge of PHP, SOAP, and HTML
- Eclipse is installed on your computer, and familiarity with using it

Code files

The GettingStartedWithWebServices_Solution.zip file includes the solutions for this tutorial. To skip following the steps, download and run the solution file to see the completed application. Before running the file, add the authentication information and your API key to the solution file. For information, see the Getting Started section.

- Unzip the GettingStartedWithWebServices_Solution.zip file under the Projects directory in your Eclipse workspace.
- Select File > Import > General > Existing Projects into Workspace.
- Click Finish.
- Run the code to see the completed application.

Setting up Eclipse

The following steps explain how to create and set up a project in Eclipse.

Creating a PHP project and file in Eclipse

- Launch Eclipse and select File > New > PHP Project to create a project.
- Enter GettingStartedWithWebServices under Project name.
- Select Finish to create the project.
- In Eclipse, select the project folder and choose File > New > PHP File.
- Name the file GettingStartedWithWebServices.
- Click Finish.

Setting up your PHP environment

The SOAP protocol is used to exchange information with the Bullhorn SOAP API. Ensure that you are using the SOAP extension with PHP. PHP 5 and later include the SOAP extension.

Using the SoapClient class

The `SoapClient` class provides a client for SOAP servers. It creates an object according to the information in the WSDL document. The `SoapClient` object has methods that correspond to the methods of the web service. You can then interact with the web service as though it were a local PHP class. For the details about this class, see **http://php.net/manual/en/class.soapclient.php** .
To verify that the soap client is installed, run a script with <?php phpinfo() ?>. The output includes "Soap Client: enabled".
In this section, you instantiate a `SoapClient` object.
Instantiate the `SoapClient` object, passing in the URL for the Bullhorn SOAP API, and parameters to set trace on and to specify the SOAP version. The trace output will be helpful for debugging.

```
$params = array(
'trace'  => 1,
'soap_version'  => SOAP_1_1
);
  $BHclient = new SoapClient("https://api.bullhornstaffing.com/webservices-
1.1/?wsdl",$params);
```

Note: At the time of publication, the latest version of the Bullhorn SOAP API is version 1.1.

Getting started

To develop applications with the Bullhorn web services API, you require a username, password, and API key. If you don't want to use your own, contact Bullhorn support to provide an API user account.

Note: Developers who work directly for Bullhorn customers get API access by contacting Bullhorn Support. After the APIs are enabled for a Bullhorn client, a client administrator can generate a customer-specific API key by going to Tools > BH Connect > Web Services API. If a key does not already exist, , click **Add Account** to create one.

Understanding how the Bullhorn SOAP API works

The sample PHP client application shows the required steps for creating a session between the client application and the Bullhorn system, and demonstrates the invocation and subsequent handling of some API calls
You program the sample application to perform the following tasks:

- Create a session to the Bullhorn web service by using your credentials.
- Build a basic query using the `dtoQuery` class.
- Retrieve a single instance of the Candidate DTO.
- Retrieve multiple instances of the Candidate DTO.
- Publish the results on the console.

Creating a session

You must create a session for all calls to the Bullhorn SOAP API. To create a session to the Bullhorn system in the web application, you require a username, password, and API key. In this section, you define the authentication mechanism for access, and create a session to the Bullhorn service.

1. In the GettingStartedWithWebServices.php file, define three variables. Bullhorn support provides these values, as described in the Getting Started section.

```
$username = "yourusername";
$password =  "yourpassword";
$apiKey = "yourapikey";
```

2. Create an empty class and assign the `username`, `password`, and `apiKey` properties to this class.

```
3.  $session_request = new stdClass;
4.  $session_request->username =  $username;
5.  $session_request->password =  $password;
```

```
$session_request->apiKey = $apiKey;
```

6. Start an API session by invoking the `startSession()` method in the `BHclient` SoapClient object. The `startSession()` method takes the `username`, `password`, and `apiKey` arguments. The method returns a session object in the response. Assign the session object to the `$API_currentSession` variable.

```
$API_session =  $BHclient->startSession($session_request);
$API_currentSession =  $API_session->return;
```

Tip: Refreshing sessions
Sessions have a short expiration and must be refreshed with each call to the Bullhorn SOAP API. Each call to the Bullhorn SOAP API requires a valid session object, and each response returns a new session object. Store that new session object in the current session variable so you always have a recently refreshed value.
Use the session object that is returned as part of the response in the next call that is made to the API, because previous session objects expire in 5 minutes.

Building a basic query

Tip: Using the Bullhorn SOAP API query operation
One of the most powerful operations in the Bullhorn web services APIs is the query operation. This operation allows you to construct SQL-like queries for a particular type of entity.
The Bullhorn query operation is built on top of Hibernate, an object-relational mapping tool that exposes relational data as a series of objects. The Hibernate Query Language (HQL) is based on standard SQL. The query operation exposes a subset of the operations supported by HQL.

To execute a query, you must construct a SOAP query Data Transfer Object (DTO) and then pass it as an argument when you call the `query` operation. The two most important fields in the query DTO are the `entityName` property, which specifies the name of the entity you are querying, and the `where` property, which contains the where clause for your query. In the where property, you specify a single parameter or create a more complex query by using `AND`, `OR`, or `NOT`.
In this section, you create a query that returns the Candidate list with at most 10 candidates that are not deleted.

Note: For referential integrity reasons, records in the Bullhorn system are never deleted. There is an `isDeleted`property on all records that you use to mark the record for deletion.

1. Create an array that contains the SQL-like properties that are executed by the Bullhorn server.

```
$query_array = array( );
```

2. In the `$query_array`, set the `entityName` property as `Candidate`.

```
'entityName' => 'Candidate',
```

3. In the `$query_array`, set the `maxResults` property equal to `10`.

```
'maxResults' => 10,
```

4. In the `$query_array`, set the `where` clause so that the query checks that the Candidate has not been deleted.

```
'where' => 'isDeleted=0',
```

Note: The where clause is a SQL-like string that is executed by the Bullhorn server. For this tutorial, the query only checks to ensure that the Candidate has not been deleted, but you can query on any of the properties exposed on the DTO. For a full list of Candidate properties, see the **reference documentation.**

5. In the `$query_array`, set the `parameters` property equal to an empty array. The `parameters` are a way of externalizing parts of your query so that you can bind values to variables. This property is optional.

```
'parameters' => array()
```

The `$query_array` appears as follows:

```
$query_array = array(
  'entityName'  => 'Candidate',
  'maxResults'  => 10,
  'where'  => 'isDeleted=0',
  'parameters'  => array()
);
```

6. The `SoapVar` class is used to set a custom data type. Cast the `$query_array` to the `dtoQuery` type which the API recognizes. The parameters for the `SoapVar` class include the data to pass in, the encoding ID, the encoding type name, and the encoding type namespace.

```
7.  $SOAP_query = new SoapVar($query_array,  SOAP_ENC_OBJECT, "dtoQuery",

"http://query.apiservice.bullhorn.com/");
```

For more information about the `SoapVar` class, see the **Using the SoapVar class to cast a data type** section.

8. Add the `$API_currentSession` and the `$SOAP_query` parameters into an array named `$request_array`. These parameters are passed into the `SoapClient` class `query()` method.

```
$request_array = array (
  'session'  => $API_currentSession,
  'query'  => $SOAP_query
 );
```

9. Use the `SoapVar` class to cast the `$request_array` to the `query` type that the API recognizes.

```
$SOAP_request = new SoapVar($request_array,  SOAP_ENC_OBJECT, "query",
 "http://query.apiservice.bullhorn.com/");
```

10. Use the `SoapClient` class to access the `query()` method in the Bullhorn SOAP API. Pass in the `$SOAP_request` as the argument. The method returns a SOAP response that is stored in the `queryResult` variable. The `ApiQueryResult`class, in the `com.bullhorn.apiservice.result` package stores the query results.

```
try{
  $queryResult = $BHclient->query($SOAP_request);
  } catch  (SoapFault $fault) {
     echo $BHclient->__getLastResult();
     die($fault->faultstring);
  }
```

Tip: Verify results

Run the debugger or use the `var_dump` command to see the `queryResult` variable with the returned data. Notice that the result contains a list of IDs. Each of these IDs corresponds to an instance of a candidate in Bullhorn. There are several operations in Bullhorn that return a list of IDs, including `getAssociationIds()` and `getEntityNotes()`methods. When using these operations, usually you then retrieve the specific instance data by using the `find()` or `findMultipe()` methods, as explained in the next section.

Retrieving instances of DTOs

After the Bullhorn system returns the results of the query in the `$queryResult` variable, you can extract the ID nodes of candidates and use each ID to retrieve a single instance of a DTO or a list of DTOs.

Retrieving a single DTO by using the find() method

The `find()` method in the Bullhorn SOAP API lets you to retrieve an instance of the DTO. It needs a session instance, the entity name, and the primary key (ID of the entity instance) to retrieve a specific record.

1. Create a `foreach` loop to access individual ID nodes in the `$queryResult` object.

```
foreach  ($queryResult->return->ids as $value){
  }
```

2. In the loop, use the `SoapVar` class to caste each ID's `$value` to an integer type.

```
$findId = new SoapVar($value, XSD_INTEGER, "int",
"http://www.w3.org/2001/XMLSchema");
```

3. In the loop, create an array that contains the current session variable, `$API_currentSession`, the `entityName` set to `'Candidate'`, and the current `$findId`.

```
$find_request = array (
  'session'  => $API_currentSession,
  'entityName'  => 'Candidate',
  'id' => $findId
);
```

4. In the loop, use the `SoapClient` class to access the `find()` method in the Bullhorn SOAP API to retrieve an instance of the DTO. The `find()` method takes the session, entity name and the ID of the `Candidate` object as arguments and returns the result as a DTO object in the `ApiFindResult` class. The DTO object returned in `$findResult`, is typed as a CandidateDto object.

```
try{
  $findResult = $BHclient->find($find_request);
  }catch  (SoapFault $fault) {
  echo $BHclient->__getLastResult();
  die($fault->faultstring);
  }
```

Publishing the results of the find() operation on console

The `candidate` DTO object properties can be displayed on the console as follows. For a complete list of properties, see the candidate DTO structure. In the previous loop, add the following `print` statements after the `try` block.

```
print "\nCandidate name: ".$findResult->return->dto->name;
print "\nOccupation: ".$findResult->return->dto->occupation;
print "\nStatus: ".$findResult->return->dto->status;
```

Retrieving a list of DTOs using findMultiple() method

You can also use the `findMultiple()` method to retrieve several instances of the DTO together. If you know you will need to fetch more than 2-3 DTOs, it is more efficient to use `findMultiple()` as it will reduce the number of round trips required to get the data. However, it will also increase the size of the responses you receive.

The `findMultiple()` method takes the session, entity name and an array of up to 20 ID nodes as arguments and returns the result as an array of `dto` objects in the `apiFindMultipleResult` class

1. Create an array named `$findId_array` that is populated with an XML node for each of the candidate `ids` typed as an integer.

```
$findId_array = array();
foreach  ($queryResult->return->ids as $value){
  $findId = new SoapVar($value, XSD_INTEGER, "int",
  "http://www.w3.org/2001/XMLSchema");
  $findId_array[] = $findId;
  }
```

2. Create an array that contains the current session variable, `$API_currentSession`, the `entityName` set to `'Candidate'`, and the `$findId_array`.

```
$find_request = array (
  'session'  => $API_currentSession,
  'entityName'  => 'Candidate',
  'ids' =>  $findId_array
);
```

3. Use the `SoapClient` class to access the `findMultiple()` method in the Bullhorn SOAP API to return an array of DTO objects.

```
try{
  $findResult = $BHclient->findMultiple($find_request);
  }catch  (SoapFault $fault) {
  echo $BHclient->__getLastResult();
  die($fault->faultstring);
  }
```

Publishing the results of the findMultiple() operation on console

1. To access individual DTOs, loop over the `dtos` array within the `$findResult` object.

```
foreach ($findResult->return->dtos  as $value){
  print  "\nCandidate name: ".$value->name;
  print  "\nOccupation: ".$value->occupation;
  print  "\nStatus: ".$value->status;
  print  "\n";
  }
```

Publishing the results on a web page

To view the results of the data retrieved from a Bullhorn system, create an HTML table and populate it with the candidate DTO data.

Creating a table for the Candidate IDs

1. Embed all the PHP code that you created in this tutorial in HTML tags.

```
<html>
<head>
<title>Getting Started with  SOAP-based Web Services</title>
<style  type="text/css">
    tr  {font-family:Arial,Helvetica,sans-serif;}
  </style>
  </head>
  <body>
    <?php
      //  PHP code goes here
  ?>
</body>
</html>
```

2. Before the closing `</body>` tag, add the following table code.

```
<table border="1" cellspacing="2"  cellpadding="2">
<tr>
<th>Candidate Id</th>
</tr>
```

```
<?php  foreach($queryResult->return->ids as $value):?>
<tr  style="text-align:center">
<td><?php echo $value;?></td>
</tr>
<?php endforeach?>
</table>
```

Creating a table for the Candidate DTO data

1. After the code in step 2 in the previous section, add the following table code.

```
<br/>
<table border="1"  cellspacing="2" cellpadding="2">
<tr>
<th>Name</th>
<th>Occupation</th>
<th>Status</th>
</tr>

<?php  foreach($findResult->return->dtos as $value):?>
<tr>
<td><?php echo $value->name;?></td>
<td><?php echo $value->occupation;?></td>
<td><?php echo $value->status;?></td>
</tr>
<?php endforeach?>
</table>
```

Running the completed application

Run your application to see the candidates retrieved from the Bullhorn system on the web page.

Tips for using PHP and the Bullhorn WSDL

The following tips are useful when working with PHP and the Bullhorn WSDL.

Using PHP 5.2/Apache 2.2 and later

We recommend using PHP 5.2/Apache 2.2 and later for this tutorial and working with the Bullhorn SOAP API.

Passing data to the Bullhorn WSDL interface

Use either the PHP's `stdClass` object or an array to pass parameters to the Bullhorn SOAP API. For example, the`startSession` operation must be passed an array or `stdClass` object that includes the username, password, and API key.

```
$session_request = new stdClass;
$session_request->username =  $username;
$session_request->password =  $password;
$session_request->apikey =  $apikey;

$API_session =  $BHclient->startSession($session_request);
$API_currentSession =  $API_session->return;
```

Using the SoapVar class to cast a data type

The Bullhorn WSDL is a typed interface and PHP is typeless, therefore, you use the `SoapVar` class to cast a data type. In the query example, you cast the `$query_array` to the `dtoQuery` type in the given namespace. This data type is one that the Bullhorn SOAP API can recognize. The parameters include the data being sent, the encoding ID, the encoding type name, and the encoding type namespace.

The `encoding ID` is an extension of the SOAP framework that defines how a data value is encoded in XML format. Use the `encoding ID` to set the XML Schema type attribute. SOAP uses XML Schema as its way of encoding data that uses XML. This tutorial uses one of the built-in XML Schema types, such as string, integer, or object.

```
$SOAP_query = new SoapVar($query_array,  SOAP_ENC_OBJECT, "dtoQuery",
"http://query.apiservice.bullhorn.com/");
```

Using a local copy of the WSDL and building from it

The Bullhorn WSDL might be updated over time. To have complete control over your implementation, we recommend that you download a copy of the Bullhorn WSDL and use the local version to build your application.

- Download the Bullhorn WSDL from https://api.bullhornstaffing.com/webservices-1.1/?wsdl.
- Save the file as bullhorn.wsdl under the GettingStartedWithWebServices project workspace.
- Use the local file bullhorn.wsdl within the SoapClient class.

Using WSDL cache settings

When you use WSDL, the client must load the relevant WSDL document from the server before an RPC call is made. This can take some time. To speed things up, you can use the WSDL caching feature on the PHP SOAP extension. Specify the settings in the `php.ini` file or use the `ini_set()` function. By default, the WSDL caching is turned on and caches WSDL files for one day. For the details of these settings,
see **http://www.php.net/manual/en/soap.configuration.php**.