# PHP Demo

Scott Hurrey

{% assign sluggedName = page.name | replace: '.md' %}

modified: {{ page.last_modified_at | date: '%b-%d-%y' }}

## Demo using PHP

The rest demo script demonstrates authenticating a REST application, management and use of the authorization token, and creating, updating, discovering, and deleting supported Learn objects.

#### Prerequisites

- You must register a developer account and application in the Developer Portal
- You must register your application in Learn
- You must also configure the script as outlined in the README for the project

This PHP command line Application allows you to:

- Authenticate
- Create, Read, and Update a Data Source
- Create, Read, and Update a Term
- Create, Read, and Update a Course
- Create, Read, and Update a User
- Create, Read, and Update a Membership
- Delete created objects in reverse order of create - membership, user, course, term, datasource.

All generated output is sent to the terminal.

**This is not meant to be a PHP tutorial. It will not teach you to write code in PHP. It will, however, give a Developer familiar with PHP the knowledge necessary to build a Web Services integration.**

#### Assumptions

This help topic assumes the Developer:

- is familiar with PHP
- has installed PHP and the HTTP_Request2 PHP Library
- has obtained a copy of the source code and built it in conjunction with the project README.md file.
- has a REST-enabled Learn instance.

#### Code Walkthrough

To build an integration with the Learn REST Web Services, regardless of the programming language of choice, can really be summed up in two steps:

1. Use the Application Key and Secret to obtain an OAuth 2.0 access token, as described in the Basic Authentication document.
2. Call the appropriate REST endpoint with the appropriate data to perform the appropriate action.

### Authorization and Authentication

The REST Services rely on OAuth 2.0 Bearer Tokens for authentication. A request is made to the token endpoint with a Basic Authorization header containing the base64-encoded key:secret string as its key. The token service returns a JSON object containing the Access Token, the Token Type, and the number of seconds until the token expires. The token is set to expire after one hour, and subsequent calls to retrieve the token will return the same token with an updated expiry time until such time that the token has expired. There is no refresh token and currently no revoke token method.

The PHP code handles this in `classes/Rest.class.php`:

```
public function authorize() {
       $constants = new Constants();
       $token = new Token();
       $request = new HTTP_Request2($constants->HOSTNAME . $constants->AUTH_PATH,
 ↪  HTTP_Request2::METHOD_POST);
       $request->setAuth($constants->KEY, $constants->SECRET, HTTP_Request2::AUTH_BASIC);
       $request->setBody('grant_type=client_credentials');
       $request->setHeader('Content-Type', 'application/x-www-form-urlencoded');
       try {
            $response = $request->send();
            if (200 == $response->getStatus()) {
                 print " Authorize Application...\n";
                 $token = json_decode($response->getBody());
            } else {
                 print 'Unexpected HTTP status: ' . $response->getStatus() . ' ' .
                      $response->getReasonPhrase();
                 $BbRestException = json_decode($response->getBody());
                 var_dump($BbRestException);
            }
       } catch (HTTP_Request2_Exception $e) {
            print 'Error: ' . $e->getMessage();
       }
       return $token;
  }
```

The JSON response is serialized into the Token object, and you may then retrieve those values from that object.

### Calling Services

The individual service calls are handled by the classes/Rest.class.php file. Each operation and object combination has its own method. Each of these methods creates the JSON body by instantiating the appropriate model from the classes directory when necessary, and then generates the appropriate HTTP Request, ships it to Learn, and serializes the JSON response back into the appropriate model.

End points are generally defined as `/learn/api/public/v1/<objecttype>/<objectId>`. Object ID can be either the pk1, like `_1_1`, or as the batchuid. This value should be prepended by externalId:, like `externalId:test101`.

For example, to retrieve a course by the pk1 `_1_1`, you would call **GET /learn/api/public/v1/courses/__1__1**. To retrieve by the batchuid `test101`, you would call **GET /learn/api/public/v1/courses/externalId:test101.**

Create is sent to Learn as a HTTP POST message with a JSON body that defines the object. The endpoint should omit the objectId, as this will be generated on creation.

Read is sent to Learn as a HTTP GET message with an empty body. The endpoint should include the objectId being retrieved.

Update is sent to Learn as a HTTP PATCH message with a JSON body that defines the object. The endpoint should include the objectId being updated.

Delete is sent to Learn as a HTTP DELETE message with empty body. The endpoint should include the objectId being deleted.

### Datasources

Datasources are handled in `classes/Rest.class.php`.

#### Create

```
public function createDatasource($access_token) {
  $constants = new Constants();
  $datasource = new Datasource();

  $request = new HTTP_Request2($constants->HOSTNAME . $constants->DSK_PATH,
   ↪  HTTP_Request2::METHOD_POST);
  $request->setHeader('Authorization', 'Bearer ' . $access_token);
  $request->setHeader('Content-Type', 'application/json');
  $request->setBody(json_encode($datasource));

  try {
      $response = $request->send();

      if (201 == $response->getStatus()) {
          print "\n Create Datasource...\n";
          $datasource = json_decode($response->getBody());
      } else {
          print 'Unexpected HTTP status: ' . $response->getStatus() . ' ' .
 ↪  $response->getReasonPhrase();
          $BbRestException = json_decode($response->getBody());
          var_dump($BbRestException);
      }
  } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
  }

  return $datasource;
}
```

#### Read

```
public function readDatasource($access_token, $dsk_id) {
  $constants = new Constants();
  $datasource = new Datasource();

  $request = new HTTP_Request2($constants->HOSTNAME . $constants->DSK_PATH . '/' . $dsk_id,
 ↪  HTTP_Request2::METHOD_GET);
  $request->setHeader('Authorization', 'Bearer ' . $access_token);

  try {
    $response = $request->send();

    if (200 == $response->getStatus()) {
      print "\n Read Datasource...\n";
      $datasource = json_decode($response->getBody());
    } else {
      print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
 ↪  .  $response->getReasonPhrase();
      $BbRestException = json_decode($response->getBody());
      var_dump($BbRestException)
    }
  } catch (HTTP_Request2_Exception $e) {
    print 'Error: ' . $e->getMessage();
  }

  return $datasource;
}
```

**Update**

```
public function updateDatasource($access_token, $dsk_id) {
  $constants = new Constants();
  $datasource = new Datasource();

  $datasource->id = $dsk_id;

  $request = new HTTP_Request2($constants->HOSTNAME . $constants->DSK_PATH . '/' . $dsk_id,
   ↪ 'PATCH');
  $request->setHeader('Authorization', 'Bearer ' . $access_token);
  $request->setHeader('Content-Type', 'application/json');
  $request->setBody(json_encode($datasource));

  try {
    $response = $request->send();

    if (200 == $response->getStatus()) {
      print "\n Update Datasource...\n";
      $datasource = json_decode($response->getBody());
    } else {
      print 'Unexpected HTTP status: ' . $response->getStatus() . ' ' .
 ↪  $response->getReasonPhrase();
      $BbRestException = json_decode($response->getBody());
      var_dump($BbRestException);
    }
  } catch (HTTP_Request2_Exception $e) {
    print 'Error: ' . $e->getMessage();
  }

  return $datasource;
}
```

**Delete**

```
public function deleteDatasource($access_token, $dsk_id) {
  $constants = new Constants();

  $request = new HTTP_Request2($constants->HOSTNAME . $constants->DSK_PATH . '/' . $dsk_id,
   ↪ HTTP_Request2::METHOD_DELETE);
  $request->setHeader('Authorization', 'Bearer ' . $access_token);
  $request->setHeader('Content-Type', 'application/json');

  try {
    $response = $request->send();

    if (204 == $response->getStatus()) {
      print "Datasource Deleted";
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' ' .
    ↪  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
        return FALSE;
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
      return FALSE;
    }

    return TRUE;
  }
```

### Terms

Terms are handled in `classes/Rest.class.php`.

### Create

```
  public function createTerm($access_token, $dsk_id) {
    $constants = new Constants();
    $term = new Term();

    $term->dataSourceId = $dsk_id;
    $term->availability = new Availability();

    $request = new HTTP_Request2($constants->HOSTNAME . $constants->TERM_PATH,
     ↪  HTTP_Request2::METHOD_POST);
    $request->setHeader('Authorization', 'Bearer ' . $access_token);
    $request->setHeader('Content-Type', 'application/json');
    $request->setBody(json_encode($term));

    try {
      $response = $request->send();

      if (201 == $response->getStatus()) {
        print "\n Create Term...\n";
        $term = json_decode($response->getBody());
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
    ↪   .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
    }

    return $term;
  }
```

### Read

```
  public function readTerm($access_token, $term_id) {
    $constants = new Constants();
    $term = new Term();
    $request = new HTTP_Request2($constants->HOSTNAME . $constants->TERM_PATH . '/' .
     ↪  $term_id, HTTP_Request2::METHOD_GET);
    $request->setHeader('Authorization', 'Bearer ' . $access_token);

    try {
      $response = $request->send();

      if (200 == $response->getStatus()) {
        print "\n Read Term...\n";
        $datasource = json_decode($response->getBody());
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
   ↪    .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
    }

    return $term;
  }
```

**Update**

```
  public function updateTerm($access_token, $dsk_id, $term_id) {
    $constants = new Constants();
    $term = new Term();

    $term->id = $term_id;
    $term->dataSourceId = $dsk_id;

    $request = new HTTP_Request2($constants->HOSTNAME . $constants->TERM_PATH . '/' .
     ↪  $term_id, 'PATCH');
    $request->setHeader('Authorization', 'Bearer ' . $access_token);
    $request->setHeader('Content-Type', 'application/json');
    $request->setBody(json_encode($term));

    try {
      $response = $request->send();

      if (200 == $response->getStatus()) {
        print "\n Update Term...\n";
        $datasource = json_decode($response->getBody());
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
   ↪    .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
    }

    return $term;
    }
```

### Delete

```
    public function deleteTerm($access_token, $term_id) {
      $constants = new Constants();

      $request = new HTTP_Request2($constants->HOSTNAME . $constants->TERM_PATH . '/' .
       ↪  $term_id, HTTP_Request2::METHOD_DELETE);
      $request->setHeader('Authorization', 'Bearer ' . $access_token);
      $request->setHeader('Content-Type', 'application/json');

      try {
        $response = $request->send();

        if (204 == $response->getStatus()) {
          print "Term Deleted";
        } else {
          print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
     ↪  .  $response->getReasonPhrase();
          $BbRestException = json_decode($response->getBody());
          var_dump($BbRestException);
          return FALSE;
        }
      } catch (HTTP_Request2_Exception $e) {
        print 'Error: ' . $e->getMessage();
        return FALSE;
      }

      return TRUE;
    }
```

### Course

Courses are handled in `classes/Rest.class.php`.

### Create

```
    public function createCourse($access_token, $dsk_id, $term_id) {
      $constants = new Constants();
      $course = new Course();

      $course->dataSourceId = $dsk_id;
      $course->termId = $term_id;
      $course->availability = new Availability();

      $request = new HTTP_Request2($constants->HOSTNAME . $constants->COURSE_PATH,
       ↪  HTTP_Request2::METHOD_POST);
      $request->setHeader('Authorization', 'Bearer ' . $access_token);
      $request->setHeader('Content-Type', 'application/json');
      $request->setBody(json_encode($course));

      try {
        $response = $request->send();

        if (201 == $response->getStatus()) {
          print "\n Create Course...\n";
          $course = json_decode($response->getBody());
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
    ↪   .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
    }

    return $course;
  }
```

**Read**

```
  public function readCourse($access_token, $course_id) {
    $constants = new Constants();
    $course = new Course();

    $request = new HTTP_Request2($constants->HOSTNAME . $constants->COURSE_PATH . '/' .
     ↪  $course_id, HTTP_Request2::METHOD_GET);
    $request->setHeader('Authorization', 'Bearer ' . $access_token);

    try {
      $response = $request->send();

      if (200 == $response->getStatus()) {
        print "\n Read Course...\n";
        $course = json_decode($response->getBody());
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
    ↪   .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
    }

    return $course;
  }
```

**Update**

```
  public function updateCourse($access_token, $dsk_id, $course_id, $course_uuid,
    ↪ $course_created) {
    $constants = new Constants();
    $course = new Course();

    $course->id = $course_id;
    $course->uuid = $course_uuid;
    $course->created = $course_created;
    $course->dataSourceId = $dsk_id;

    $request = new HTTP_Request2($constants->HOSTNAME . $constants->COURSE_PATH . '/' .
     ↪  $course_id, 'PATCH');
    $request->setHeader('Authorization', 'Bearer ' . $access_token);
    $request->setHeader('Content-Type', 'application/json');
    $request->setBody(json_encode($course));

    try {
      $response = $request->send();

      if (200 == $response->getStatus()) {
        print "\n Update Course...\n";
        $course = json_decode($response->getBody());
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
↪   .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
    }

    return $course;
  }
```

**Delete**

```
  public function deleteCourse($access_token, $course_id) {
    $constants = new Constants();

    $request = new HTTP_Request2($constants->HOSTNAME . $constants->COURSE_PATH . '/' .
     ↪  $course_id, HTTP_Request2::METHOD_DELETE);
    $request->setHeader('Authorization', 'Bearer ' . $access_token);
    $request->setHeader('Content-Type', 'application/json');

    try {
      $response = $request->send();

      if (204 == $response->getStatus()) {
        print "Course Deleted";
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
↪   .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
        return FALSE;
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
      return FALSE;
    }

    return TRUE;
  }
```

**Users**

Users are handled in `classes/Rest.class.php`.

**Create**

```
public function createUser($access_token, $dsk_id) {
  $constants = new Constants();
  $user = new User();

  $user->dataSourceId = $dsk_id;
  $user->availability = new Availability();
  $user->name = new Name();
  $user->contact = new Contact();

  $request = new HTTP_Request2($constants->HOSTNAME . $constants->USER_PATH,
   ↪  HTTP_Request2::METHOD_POST);
  $request->setHeader('Authorization', 'Bearer ' . $access_token);
  $request->setHeader('Content-Type', 'application/json');
  $request->setBody(json_encode($user));

  try {
    $response = $request->send();

    if (201 == $response->getStatus()) {
      print "\n Create User...\n";
      $user = json_decode($response->getBody());
    } else {
      print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
↪   .  $response->getReasonPhrase();
      $BbRestException = json_decode($response->getBody());
      var_dump($BbRestException);
    }
  } catch (HTTP_Request2_Exception $e) {
    print 'Error: ' . $e->getMessage();
  }

  return $user;
}
```

**Read**

```
public function readUser($access_token, $user_id) {
  $constants = new Constants();
  $user = new User();

  $request = new HTTP_Request2($constants->HOSTNAME . $constants->USER_PATH . '/' .
   ↪  $user_id, HTTP_Request2::METHOD_GET);
  $request->setHeader('Authorization', 'Bearer ' . $access_token);

  try {
    $response = $request->send();

    if (200 == $response->getStatus()) {
      print "\n Read User...\n";
      $user = json_decode($response->getBody());
    } else {
      print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
↪   .  $response->getReasonPhrase();
      $BbRestException = json_decode($response->getBody());
      var_dump($BbRestException);
    }
  } catch (HTTP_Request2_Exception $e) {
    print 'Error: ' . $e->getMessage();
  }

  return $user;
}
```

**Update**

```
  public function updateUser($access_token, $dsk_id, $user_id, $user_uuid, $user_created) {
    $constants = new Constants();
    $user = new User();

    $user->id = $user_id;
    $user->uuid = $user_uuid;
    $user->created = $user_created;
    $user->dataSourceId = $dsk_id;

    $request = new HTTP_Request2($constants->HOSTNAME . $constants->USER_PATH . '/' .
    ↪  $user_id, 'PATCH');
    $request->setHeader('Authorization', 'Bearer ' . $access_token);
    $request->setHeader('Content-Type', 'application/json');
    $request->setBody(json_encode($user));

    try {
      $response = $request->send();

      if (200 == $response->getStatus()) {
        print "\n Update User...\n";
        $user = json_decode($response->getBody());
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
  ↪   .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
    }

    return $user;
  }
```

**Delete**

```
  public function deleteUser($access_token, $user_id) {
    $constants = new Constants();

    $request = new HTTP_Request2($constants->HOSTNAME . $constants->USER_PATH . '/' .
    ↪  $user_id, HTTP_Request2::METHOD_DELETE);
    $request->setHeader('Authorization', 'Bearer ' . $access_token);
    $request->setHeader('Content-Type', 'application/json');

    try {
      $response = $request->send();

      if (204 == $response->getStatus()) {
        print "User Deleted";
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
    ↪   .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
        return FALSE;
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
      return FALSE;
    }

    return TRUE;
  }
```

### Memberships

Memberships are handled in `classes/Rest.class.php`.

### Create

```
  public function createMembership($access_token, $dsk_id, $course_id, $user_id) {
    $constants = new Constants();
    $membership = new Membership();

    $membership->dataSourceId = $dsk_id;
    $membership->availability = new Availability();

    $request = new HTTP_Request2($constants->HOSTNAME . $constants->COURSE_PATH . '/' .
    ↪  $course_id . '/users/' . $user_id, HTTP_Request2::METHOD_PUT);
    $request->setHeader('Authorization', 'Bearer ' . $access_token);
    $request->setHeader('Content-Type', 'application/json');
    $request->setBody(json_encode($membership));

    try {
      $response = $request->send();

      if (201 == $response->getStatus()) {
        print "\n Create Membership...\n";
        $membership = json_decode($response->getBody());
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
    ↪   .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
    }

    return $membership;
  }
```

### Read

```
  public function readMembership($access_token, $course_id, $user_id) {
    $constants = new Constants();
    $membership = new Membership();

    $request = new HTTP_Request2($constants->HOSTNAME . $constants->COURSE_PATH . '/' .
     ↪  $course_id . '/users/' . $user_id,  HTTP_Request2::METHOD_GET);
    $request->setHeader('Authorization', 'Bearer ' . $access_token);

    try {
      $response = $request->send();

      if (200 == $response->getStatus()) {
        print "\n Read Membership...\n";
        $membership = json_decode($response->getBody());
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
   ↪  .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
      }
    } catch (HTTP_Request2_Exception $e) {
      print 'Error: ' . $e->getMessage();
    }

    return $membership;
  }
```

**Update**

```
  public function updateMembership($access_token, $dsk_id, $course_id, $user_id,
   ↪  $membership_created) {
    $constants = new Constants();
    $membership = new Membership();

    $membership->dataSourceId = $dsk_id;
    $membership->userId = $user_id;
    $membership->courseId = $course_id;
    $membership->created = $membership_created;
    $membership->availability = new Availability();

    $request = new HTTP_Request2($constants->HOSTNAME . $constants->COURSE_PATH . '/' .
     ↪  $course_id . '/users/' . $user_id, 'PATCH');
    $request->setHeader('Authorization', 'Bearer ' . $access_token);
    $request->setHeader('Content-Type', 'application/json');
    $request->setBody(json_encode($membership));

    try {
      $response = $request->send();

      if (200 == $response->getStatus()) {
        print "\n Update Membership...\n";
        $membership = json_decode($response->getBody());
      } else {
        print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
   ↪  .  $response->getReasonPhrase();
        $BbRestException = json_decode($response->getBody());
        var_dump($BbRestException);
      }
  } catch (HTTP_Request2_Exception $e) {
    print 'Error: ' . $e->getMessage();
  }

  return $membership;
}
```

**Delete**

```
public function deleteMembership($access_token, $course_id, $user_id) {
  $constants = new Constants();

  $request = new HTTP_Request2($constants->HOSTNAME . $constants->COURSE_PATH . '/' .
   ↪  $course_id . '/users/' . $user_id, HTTP_Request2::METHOD_DELETE);
  $request->setHeader('Authorization', 'Bearer ' . $access_token);
  $request->setHeader('Content-Type', 'application/json');

  try {
    $response = $request->send();

    if (204 == $response->getStatus()) {
      print "Membership Deleted";
    } else {
      print 'Unexpected HTTP status: ' . $response->getStatus() . ' '
 ↪  .  $response->getReasonPhrase();
      $BbRestException = json_decode($response->getBody());
      var_dump($BbRestException);
      return FALSE;
    }
  } catch (HTTP_Request2_Exception $e) {
    print 'Error: ' . $e->getMessage();
    return FALSE;
  }

  return TRUE;
}
```

## Conclusion

All of the code snippets included in this document at included in a sample REST Demo PHP application available on GitHub. There is a README.html included that talks more specifically about building and running the code. Feel free to review the code and run it against a test or development Learn instance to see how it works.