Event Hooks for Non-Developers/Weekend Developers

Author: Jeff Nester
Date: 06/21/2019

Purpose: To provide a simple example of using Okta Event Hooks.

Table of Contents

Introduction

I am what I call a "weekend developer." That means with Google's help and enough time I can code almost anything. The goal of this paper is to provide a simple example of using Okta's Event Hooks that someone with a little bit of programming knowledge can follow. I have found that PHP is a simple programming language to understand. Therefore, this paper will use PHP as the programing language.

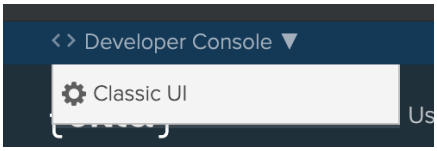
For details about Okta's Event Hooks go to:

- https://developer.okta.com/use_cases/event_hooks/ and
- https://developer.okta.com/docs/api/resources/event-hooks/#getting-started

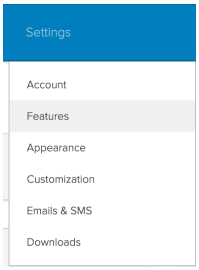
Prerequisites

In order to use Okta Event Hooks there are a few things that you will need.

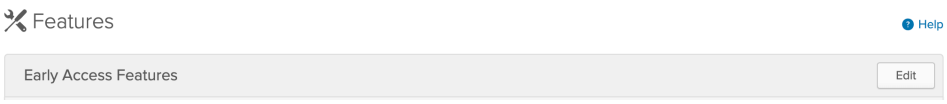
1. You need an Okta org. If you don't have one you can obtain one by clicking on "Sign Up" on https://developer.okta.com
2. You need the appropriate feature flags enabled in your org. This is done by:
    a. Log in to you Okta org
    b. Go to the Admin Page
    c. Change the view to ***Classic***. This is at the top of the page.



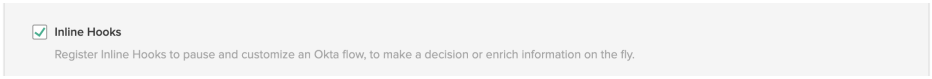
    d. Go to ***Settings*** → ***Features***



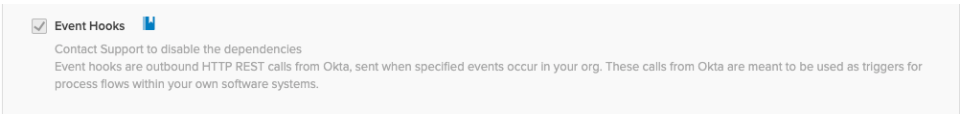
    e. Click the ***Edit*** button



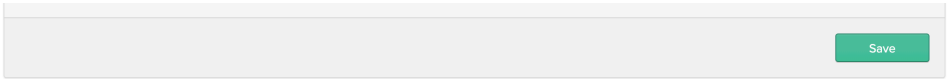
f. Request "***Inline Hooks***" by checking the box. (This is a requirement for Event Hooks)



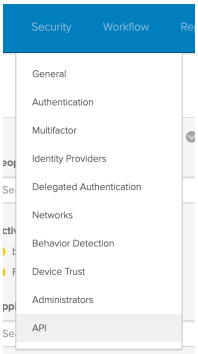
g. Request the "***Event Hooks***" Early Access Feature by checking the box



h. After the features are selected click the ***Save*** button

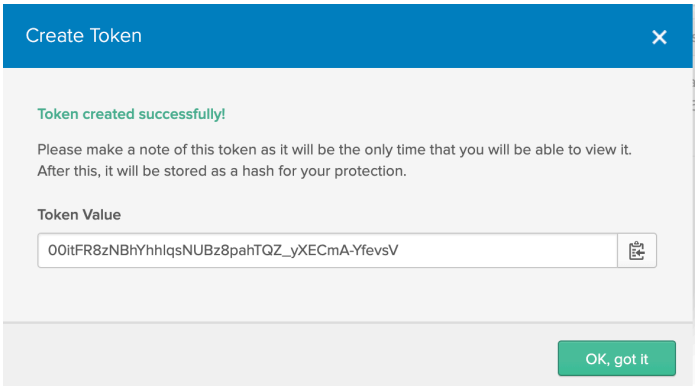


3. An API Token is required. This will be needed later in the setup of the Postman collections. To get an API Token do:
   a. Login to you Okta Org as described above and select the Classic UI
   b. Click on ***Security → API***

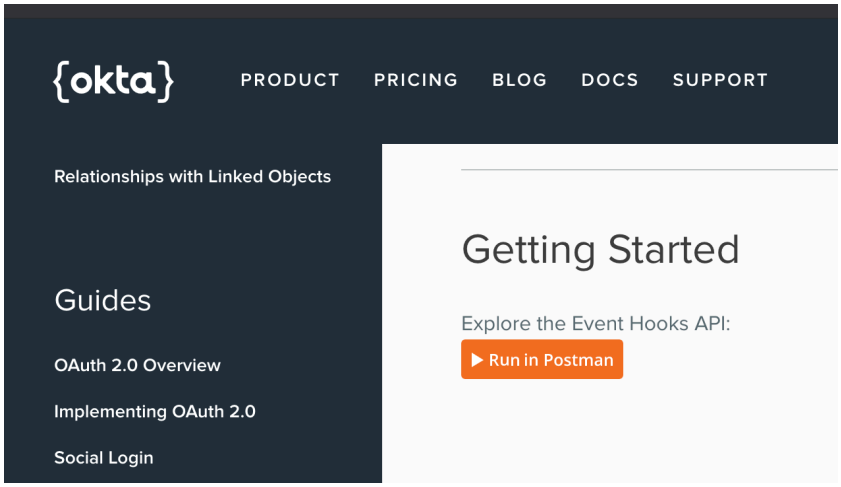


   c. Click on the ***Tokens*** Sub Tab
   d. Click on the ***Create Token*** button. This will prompt you for a name for the token.

e. Click on the green ***Create Token*** button. This will present you with your API token. You must copy and **save** this Token for later. NOTE: it cannot be re-displayed. When you click the ***"OK, got it"*** button it is no longer viewable.

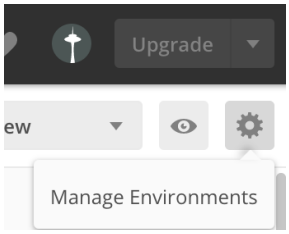


4. Install your favorite Web Server and enable PHP. (This must be accessible from the public internet. Okta will be making calls to the web site.)
5. You will need POSTMAN to configure your event and verify the event services. POSTMAN can be obtained from https://www.getpostman.com
6. Configure POSTMAN
    a. Once POSTMAN is installed download the Event Hooks Collection to POSTMAN. Click on the ***Run in Postman*** link on this page https://developer.okta.com/docs/api/resources/event-hooks/#getting-started



    b. Now we have to configure POSTMAN to use your Okta Org

c. Click on the Manage Gear at the top right



d. Click on the ***Add*** button at the bottom of the screen and give the profile a name then enter the data shown below in the screen:

VARIABLE	INITIAL VALUE	CURRENT VALUE
url	https://{yourOktaOrgURL}	
apikey	{The Token you save above}	
eventHookId	{leave blank}	

e. Click on the ***Add*** button. Then close the window.
f. Make sure that the profile is selected in the drop down to the left of the eye icon.

Deploy the Event Handler

In your PHP enabled webserver's doc folder create a folder called **events**. On a Linux server this is usually ***/var/www/html***. Don't worry about what the code does right now. We will walk through it later in this document.

Go to the **events** directory on the webserver and create index.php that contains:

```
<?php
// Name: Simple Event Hook Example
//
// Purpose: To explain how Event Hooks without getting caught up in code.
//          This is a simple PHP example that will respond to the request
//.         from an Event Hook firing.
//
// Author: Jeff Nester
//
// List the headers so you can see what is being passed. All output goes
// to the web server default logs. i.e. /var/www/log/errors
//
error_log("--> EVENT_HOOKS ----- " . "|------------- Start of index.php -----------------
-------- ");
error_log("--> EVENT_HOOKS ----- " . "| Headers: ");

// Fetch headers for display
$headers = getallheaders ();

// write headers to error log
foreach ( $headers as $key=>$val) {
   error_log("--> EVENT_HOOKS ----- |    " . $key . ": " . $val);
}
error_log("--> EVENT_HOOKS ----- " . "|-------------------------------------------------
--------");
```

```
//
// When Okta attempts to verify the service, it will send a GET request.
// When an event fires it will send a POST request. This switch statement
// determines if it is a GET (Verification) or POST (new event)
//
switch($_SERVER['REQUEST_METHOD']) {
   case 'GET':
// Verification
      error_log( "--> EVENT_HOOKS ----- | Verification Request:");
// obtain the verification code from the header. This must be returned in
// a JSON response.
      $code = $_SERVER["HTTP_X_OKTA_VERIFICATION_CHALLENGE"];

// build response
      $jsonString = array ('verification' => $code);
      $json = json_encode($jsonString);
      error_log("--> EVENT_HOOKS ----- |    Response: " . $json);

 // Send response back to Okta with verification code in JSON
      echo $json;
      error_log("--> EVENT_HOOKS ----- " . "|-----------------------------------------
--------------");
      break;

        case 'POST':
        // New Event(s)
        // Obtain the JSON from the request
        $data = json_decode(file_get_contents('php://input'), true);

        // from the data extract the events
        $events = $data['data']['events'];

        // loop through the events and print to the web server log the results
        foreach ($events as $key => $value) {
          error_log("--> EVENT_HOOKS ----- " . "| Event: " . $value["eventType"] );
         $actor = $value["actor"];
         error_log("--> EVENT_HOOKS ----- " . "|    ID:           " . $actor['id']);
         error_log("--> EVENT_HOOKS ----- " . "|    Type:         " . $actor['type']);
         error_log("--> EVENT_HOOKS ----- " . "|    Alternate ID: " .
$actor['alternateId']);
         error_log("--> EVENT_HOOKS ----- " . "|    Display Name: " .
$actor['displayName']);
         error_log("--> EVENT_HOOKS ----- " . "|-----------------------------------------
----------------");

        // Send the response - It is an empty JSON string
         echo "{}";
        }
        break;
}
?>
```

Create the Event Hook in Okta

For your code to be called a trigger must be configured in Okta. This trigger will include:

- Name
- The Event Items to trigger on
- The pointer to your event handler code including security configuration
- Security information

In this example we will trigger on two events:

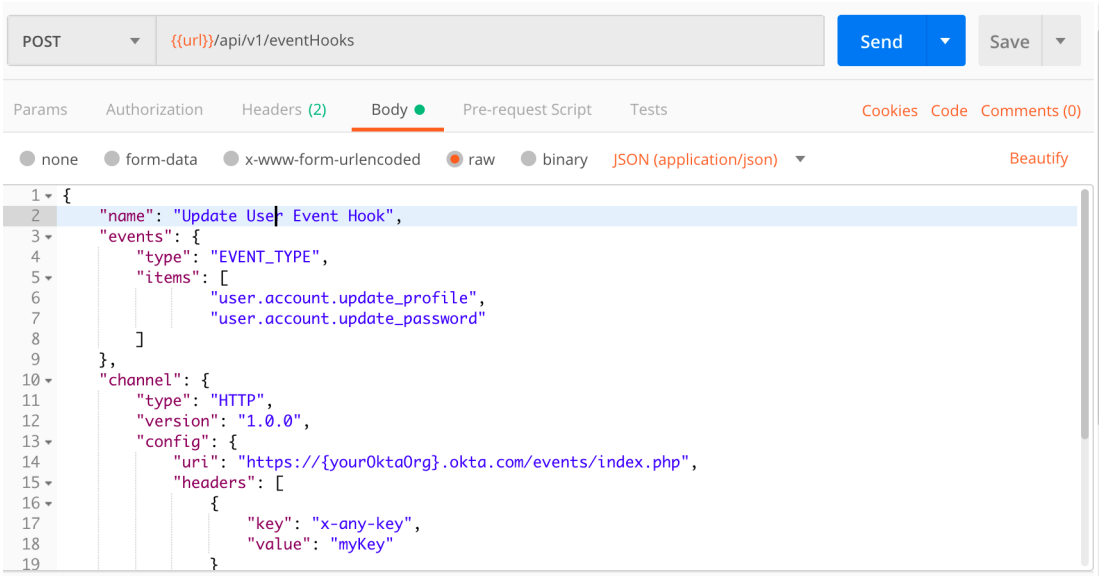
- "user.account.update_profile"
- "user.account.update_password"

The event handler I am using is written in PHP and was deployed earlier in this document. When creating an Event Hook there are two steps. First you must create the Event Hook in Okta. Then you have to verify that Okta can use the end point that was configured. This is done by Okta making a GET request to the php code that was deployed.

Create the Event Hook in Okta

Perform the following steps to configure the Event Hook:

1. In POSTMAN click on the ***Event Hooks → Create new Event Hook***
2. Click on the ***Body*** sub tab



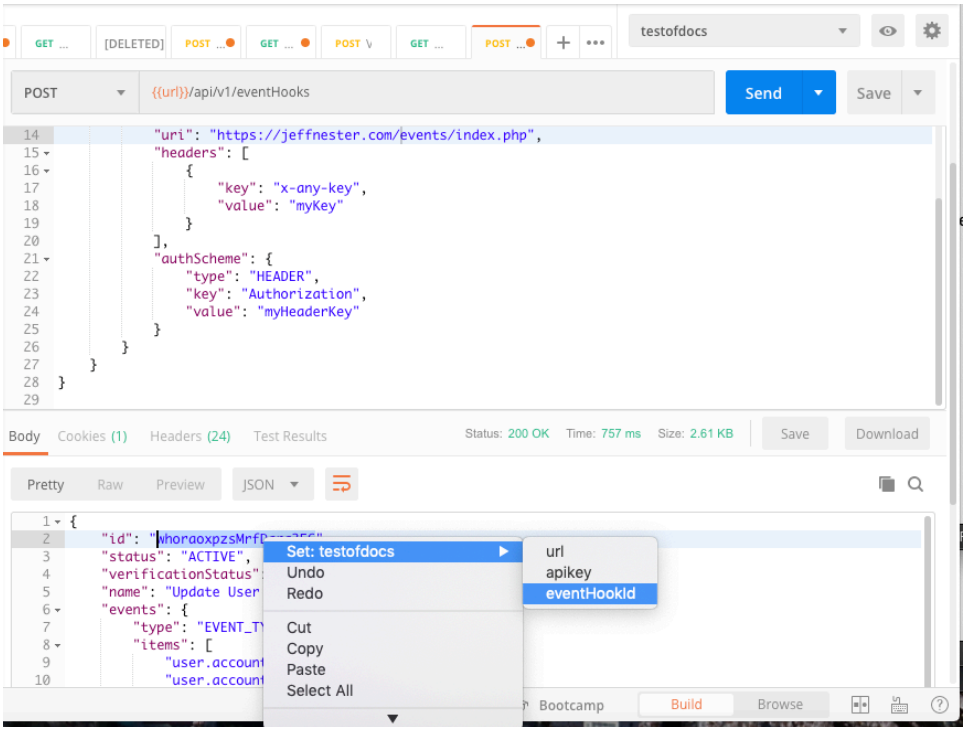
3. In the body place the following JSON. [Make sure to replace the uri with your value]

```
{
    "name": "Update User Event Hook",
    "events": {
        "type": "EVENT_TYPE",
        "items": [
                "user.account.update_profile",
                "user.account.update_password"
        ]
    },
    "channel": {
        "type": "HTTP",
        "version": "1.0.0",
        "config": {
            "uri": "https://{yourWebServerURL}/events/index.php",
            "headers": [
                {
                    "key": "x-any-key",
                    "value": "myKey"
                }
            ],
            "authScheme": {
```

```
                "type": "HEADER",
                "key": "Authorization",
                "value": "myHeaderKey"
            }
        }
    }
}
```

4. Click the ***Send*** Button to execute the creation. If it is successful you will get a 200 response and a JSON response that will contain the newly created webhook ID.

5. Next, we need to grab the id that was returned in the JSON. There is a really neat POSTMAN trick to do this. In the results Body section in Postman highlight the id value and then right mouse over the highlighted text. Like this:

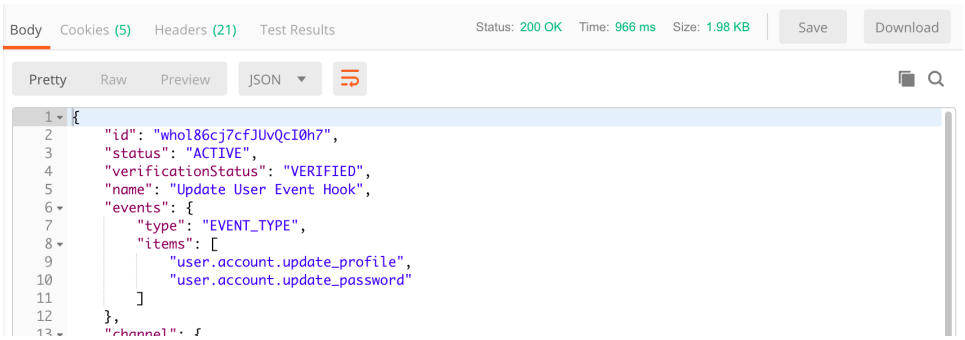


At this time, we have successfully created an Event Hook in Okta and have saved the Event Hook's id so we can use it in the next step

Verify the Created Event Hook

Perform the following steps to Verify the Event Hook:

1. In POSTMAN click on the ***Event Hooks → Verify an Event Hook***
2. Since we set the value for ***eventHookId*** in the last step there is nothing to do but click on the ***Send*** button. If it is successful you will get a 200 status and some JSON returned. It should look similar to:



Note: The PHP code doesn't actually validate the security data that is sent to it so there is no need to send any parameters to this code. In production you would never do this but for this simple code example the check was left out.

Testing the Event

Now that we have everything configured, we need to test out the Event Hook and see if it is working. The PHP code writes debug information to the web server's error log file. In apache this can be found at **/var/log/httpd/error_log**. If you are running the web server on Linux, you can watch the log in real time by doing **tail -f /var/log/httpd/error_log**.

Let's test out the Event Hook. We have registered this event to execute on update_profile and update_password. This means that if any user record in the Okta org is updated or the password changes the event will trigger and call the deployed PHP code.

Do the following to test the Event Hook:

1. Log into your Okta org as an administrator
2. Go to **Directory → People**
3. Pick a user that is Okta Mastered or create a new user
4. Modify a profile attribute
5. Look at the **error_log** file (If you were using tail -f the results would already be on the terminal screen where you executed it)

   The results should look something like this:

```
{timestamp}--> EVENT_HOOKS ----- |------------- Start of index.php -------------------------
{timestamp}--> EVENT_HOOKS ----- | Headers:
{timestamp}--> EVENT_HOOKS ----- |    Authorization: myHeaderKey
{timestamp}--> EVENT_HOOKS ----- |    x-any-key: myKey
{timestamp}--> EVENT_HOOKS ----- |    User-Agent: Okta Hook Service v1.0.0 - whol86cj7cfJUvQcI0h7
```

```
{timestamp}--> EVENT_HOOKS ----- |    Accept: application/json
{timestamp}--> EVENT_HOOKS ----- |    Content-Type: application/json
{timestamp}--> EVENT_HOOKS ----- |    Host: jeffnester.com
{timestamp}--> EVENT_HOOKS ----- |    Connection: keep-alive
{timestamp}--> EVENT_HOOKS ----- |    Content-Length: 1985
{timestamp}--> EVENT_HOOKS ----- |-------------------------------------------------------
{timestamp}--> EVENT_HOOKS ----- | Event: user.account.update_profile
{timestamp}--> EVENT_HOOKS ----- |    ID:           00u5m9em2zydezGcf0h7
{timestamp}--> EVENT_HOOKS ----- |    Type:         User
{timestamp}--> EVENT_HOOKS ----- |    Alternate ID: jeff@jeffnester.com
{timestamp}--> EVENT_HOOKS ----- |    Display Name: Jeff
{timestamp}--> EVENT_HOOKS ----- |-------------------------------------------------
```

Note: in your log the **{timestamp}** will look similar to:

```
[Mon Jun 17 13:09:54.131706 2019] [php7:notice] [pid 16742] [client 34.200.8.251:64279]
```

It was removed to make the output easier to read in Microsoft Word.

A Look at the Code

This section will walk through the code and explain what each section is doing.

All PHP programs must start with this string.

```
<?php
```

It is always great to add comments into your code. It helps the next person that looks at the code understand what it is doing. This block is just a description of what the file is intended to do.

```
// Name: Simple Event Hook Example
//
// Purpose: To explain how Event Hooks without getting caught up in code.
//          This is a simple PHP example that will respond to the request
//.         from an Event Hook firing.
//
// Author: Jeff Nester
//
// List the headers so you can see what is being passed. All output goes
// to the web server default logs. i.e. /var/www/log/errors
//
```

error_log() is the php method that is used to write information into the web server's error_log file. This section writes all of the header information that was passed to the php file into the error_log file.

```
error_log("--> EVENT_HOOKS ----- " . "|------------ Start of index.php -----------------
-------- ");
error_log("--> EVENT_HOOKS ----- " . "| Headers: ");

// Fetch headers for display
$headers = getallheaders ();

// write headers to error log
foreach ( $headers as $key=>$val) {
   error_log("--> EVENT_HOOKS ----- |    " . $key . ": " . $val);
}
error_log("--> EVENT_HOOKS ----- " . "|-------------------------------------------------
--------");
```

To verify that Okta can execute the php code when an event triggers the code must provide a verification method. If the request is a GET request, then Okta is attempting to verify the event handler and the code should respond appropriately.

```
//
// When Okta attempts to verify the service, it will send a GET request.
// When an event fires it will send a POST request. This switch statement
// determines if it is a GET (Verification) or POST (new event)
//
```

The switch statement is being used to distinguish between a verification request and an event trigger request. As stated above the HTTP Request type will be different so we will let the switch statement evaluate the REQUEST_METHOD that is passed in on the HTTP Request.

```
switch($_SERVER['REQUEST_METHOD']) {
```

If the REQUEST_METHOD is equal to “GET” then it is a verification request and we will execute the verification code.

```
   case 'GET':
// Verification
```

To make it clear in the error_log file what is happening this code writes to the error_log a message indicating that Verification process is executing.

```
      error_log( "--> EVENT_HOOKS ----- | Verification Request:");
// obtain the verification code from the header. This must be returned in
// a JSON response.
```

For the verification to be successful Okta will send a challenge code in the header variable “HTTP_X_OKTA_VERIFICATION_CHALLENGE” that must be returned to Okta in the body of the results as a JSON string

So first we get the challenge code from the header

```
      $code = $_SERVER["HTTP_X_OKTA_VERIFICATION_CHALLENGE"];
```

Next take the code and build a JSON string to send back. It must be in the form of **{ “verification”: “challenge code”}** The next few lines of code will create this string and write it to the error_log file so that you can see what is happening.

```
// build response
      $jsonString = array ('verification' => $code);
      $json = json_encode($jsonString);
      error_log("--> EVENT_HOOKS ----- |    Response: " . $json);
```

Now reply to Okta with the correctly formatted response. This is done by simply writing the string using echo.

```
// Send response back to Okta with verification code in JSON
     echo $json;
```

The code will then write to the error_log file to indicate that the event verification code has completed execution.

```
     error_log("--> EVENT_HOOKS ----- " . "|-----------------------------------------
--------------");
```

The break statement indicates this is the end of the block of code in the switch statement.

```
     break;
```

The case statement is checking to see if the REQUEST_METHOD is equal to POST. A POST request indicates this is an event that has been triggered.

```
case 'POST':
// New Event(s)
```

To begin processing the event that was triggered we must obtain the JSON string that was sent from Okta to the PHP code. This is done as using a combination of the two statements: file_get_contents and json_decode.

```
// Obtain the JSON from the request
$data = json_decode(file_get_contents('php://input'), true);
```

The JSON contains a bunch of information but for our event handler we only care about the events section. This can be extracted from the JSON by doing:

```
// from the data extract the events
$events = $data['data']['events'];
```

In our example we don't really do much with the data that we received. This code simply writes this data to the error_log file so that you can see the data. Normally this code would perform some action because this event fired. My action is to log the information to the error_log file.

```
// loop through the events and print to the web server log the results
foreach ($events as $key => $value) {
  error_log("--> EVENT_HOOKS ----- " . "| Event: " . $value["eventType"] );
 $actor = $value["actor"];
 error_log("--> EVENT_HOOKS ----- " . "|    ID:           " . $actor['id']);
 error_log("--> EVENT_HOOKS ----- " . "|    Type:         " . $actor['type']);
 error_log("--> EVENT_HOOKS ----- " . "|    Alternate ID: " . $actor['alternateId']);
 error_log("--> EVENT_HOOKS ----- " . "|    Display Name: " . $actor['displayName']);
 error_log("--> EVENT_HOOKS ----- " . "|-----------------------------------------");
```

Okta is expecting a response once the code is completed. The response needs to be a 200 response with an empty JSON string returned. This is done using the echo command

```
            // Send the response - It is an empty JSON string
             echo "{}";
            }
```

As before break indicates that there is no more work to be done in this section of the switch statement.

```
            break;
    }
```

PHP programs have to start with <?php and they end with ?>

```
    ?>
```

Conclusion

At this point you have successfully:

- created an event hook in Okta
- verified the hook
- tested it
- understand how to perform the action using PHP

The concept of how event hooks work is the same regardless of the programming language that you use. The code will be a bit different, but the steps will be the same.