# Linking OneID with existing accounts

## Introduction

Integrating with OneID Sign In requires the storage of a unique GUID that is unique to the user and your system. Depending on when the user performs the action to connect their OneID with your system, there may be different user experiences and logic changes. This document will discuss each scenario, and the recommended approach to implement a solution.

## Process Flow

All requests for linking an account with OneID use the login API call described here: <https://developer.oneid.com/display/docs/JavaScript+API#JavaScriptAPI-loginbutton>

OneID provides sample code that calls this API from many common web languages. This document will use the PHP SDK as reference, which is available at <https://github.com/OneID/oneid-php-sdk>.

The diagram below shows the flow of a request between your system and OneID when a sign in is requested.



This document will focus on the Callback portion of the process.

## Callback Implementation

Generally, you’ll only need to create one callback in your system to handle all the various linking events. The PHP callback is described here (found in validate.php)

|  |
| --- |
| **Code** |
| require("oneid.php");  $response = OneID\_Response();  // This function handles validation of  // the response from OneID and returns an  // array of attributes for the user.  session\_start();  if(OneID\_IsSuccess($response)) {  // here you do what you need to link the OneID with the current user.  $\_SESSION['email'] = $response['attr']['personal\_info']['email'];  $\_SESSION['OneID'] = $response['uid'];  } else {  // if it fails, clear whatever link may have been set previously  unset($\_SESSION['OneID']);  unset($\_SESSION['email']);  $\_SESSION=array\_values($\_SESSION);  }  // return the redirect to the next page  echo OneID\_Redirect('account.php',$response);  // OneID\_redirect will send the browser to the next page and pass any errorcodes from authentication and verification back to the OneID client. |

## This callback assumes you are using the AJAX validation callback. If you use the POST validation method, your method would return an HTTP result and new page instead of calling OneID\_Redirect.

The first relevant portion of this code calls OneID\_Response(). This call grabs the contents of the POST buffer of the request and sends the package to our Keychain service for validation. The keychain service abstracts away the cryptographic signature validation. It is possible to do the signature validation locally, but requires a more specialized library. Please contact us if you would like to deploy this local keychain system.

The result of this call parses the JSON result automatically and puts it into the return dictionary. In the sample code, it puts the email address and OneID unique ID in the response dictionary.

The next part of the callback makes a call to see if the OneID response was valid. This call just checks the errorcode field in the response dictionary to ensure that it is 0.

Once the response has been received and validated, you should do the logic required to link the user’s OneID to your system. Typical scenarios are described below:

## Linking Scenarios

### 1: User is not logged into your system and is not a registered user

Depending on your site, you may choose to register the user using the supplied email address and the OneID unique ID. You may add arguments to the login API function to request more attributes. The list of available attributes is available at:

<https://developer.oneid.com/display/docs/User+Attributes>.

If you choose to present a local registration screen, you should keep the OneID unique ID in the user’s session, and attach it to the user’s account when the registration has completed.

|  |
| --- |
| **Sample callback modified for linking scenarios** |
| ...  if(OneID\_IsSuccess($response)) {  if (!IS\_LOGGED\_IN) {  // here you do what you need to link the OneID with the current user.    // Look up account by UID  if (ACCOUNT\_EXISTS($response['uid'])){  LOG\_IN\_BY\_UID($response['uid'])  }  else {  if (ACCOUNT\_EXISTS($email)){  // Account exists, but is not linked. Force user to login.  // Stash the uid into the session, so we can pick up on it later.  $\_SESSION["uid"] = $response["uid"];  redirect("traditional\_login.php");  }  else {  // Can't find an account with this info, create one.  USER\_OBJ = new LOCAL\_USER\_CLASS();  USER\_OBJ.oneID = $response['uid'];  USER\_OBJ.email = $email;  USER\_OBJ.save();  }  }  } else {  $USER\_OBJ.oneID = $response['uid'];  $USER\_OBJ.save();  $redirect = “account.php”;  }  } else {  ...  }  // return the redirect to the next page  echo OneID\_Redirect($redirect,$response);  // OneID\_redirect will send the browser to the next page and pass any errorcodes from authentication and verification back to the OneID client. |

### 2: User is not logged into your system, but is a registered user

Your site may require that a user log in to their existing account. Typical examples of this are sites that use email address as a primary account identifier, or sites that require users to have a preexisting account created via some other process (such as a financial institution).

In this scenario, you should present the user with a screen that allows them to sign in with their existing username and password one last time. Once you’ve authenticated the user, you should attach the previously presented OneID unique ID to their existing account.

### 3: User is logged into your system, and clicks on an offer to link their OneID to their account.

This scenario does not require any further user interaction. Just add the user’s OneID to their account and save the user object back to your data store. The user should be notified that their account is now linked to their OneID.

## Other Resources

Visit developer.oneid.com to find other implementations and package solutions for your platform. If you need additional support, contact us at [support@oneid.com](mailto:support@oneid.com).