Login with Amazon

Developer Guide for Websites

Login with amazon

Login with Amazon: Developer Guide for Websites

Contents

Welcome

This is the *Login with Amazon Developer Guide for Websites*. This guide contains conceptual information about the Login with Amazon web service, as well as information about how to use Login with Amazon in your website.

Login with Amazon is a web service that enables Amazon customers to login to your website or mobile app using their Amazon credentials. Once they have logged in, your app can access some information from their Amazon profile.

How Do I...?

See the following table for links to information on how to work with Login with Amazon.

How Do I...	Relevant Resources
Learn more about the business case for Login with Amazon	Understanding Login with Amazon
Learn how Login with Amazon works	Login with Amazon Conceptual Overview
Get started with Login with Amazon	Login with Amazon Getting Started Guide for Websites
Decide whether Login with Amazon is the right choice for my application	Using Login with Amazon
Get the FAQ	Login with Amazon Frequently Asked Questions
Get help from the community of developers	Login with Amazon Discussion Forums

Understanding Login with Amazon

Topics

Login with Amazon enables Amazon customers to use their trusted Amazon account to log into websites and mobile apps.

This section shows how Login with Amazon uses access tokens to allow websites to login customers and access their customer profiles.

Login with Amazon Conceptual Overview

The conceptual overview describes how Login with Amazon allows a user to login and grant your website or access to their customer profile data. For more details on the customer experience in native mobile apps, including how your customers can skip the login screen and experience single sign-on, please see our Customer Experience Overview for Android/Fire apps, and our Customer Experience Overview for iOS apps.

The Login with Amazon process begins when a user visits your website or app (A). They click the Login with Amazon button (B) and get redirected to a login screen. Amazon provides pages (C) where the user logs in, then consents to allow your website access to their profile data. If they have already consented, they will only have to login. Amazon then redirects the user from the login screen to your website or app (D). Your website or app uses security credentials provided by Login with Amazon to access the customer profile (E) (including name and email address).

If a Login with Amazon app wants to identify a user without accessing their name and email address, they will not request profile data. In this case, the user is not presented with a consent screen after they log in.

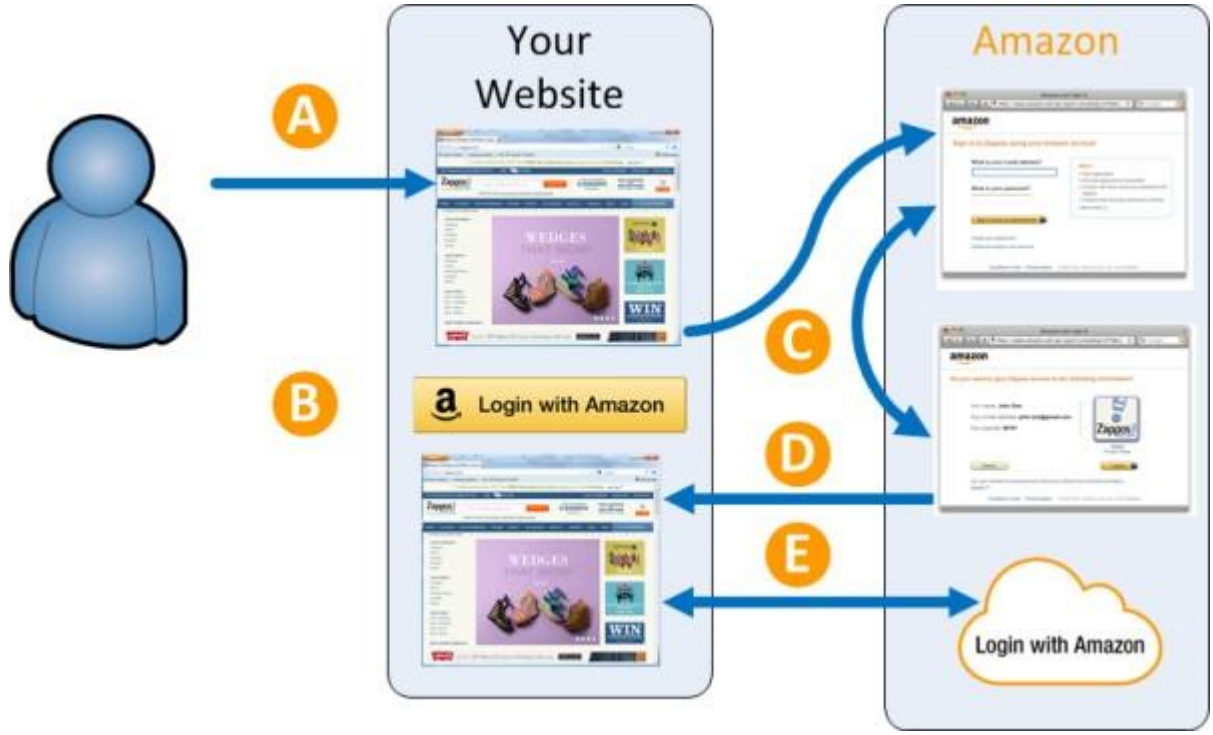


Figure 1: Login with Amazon user flow

Login with Amazon works by providing third-party websites and mobile apps (clients) with a recognizable login button that users click to sign in with their Amazon credentials. To login, users are directed to amazon.com and asked to provide their Amazon password. For example:

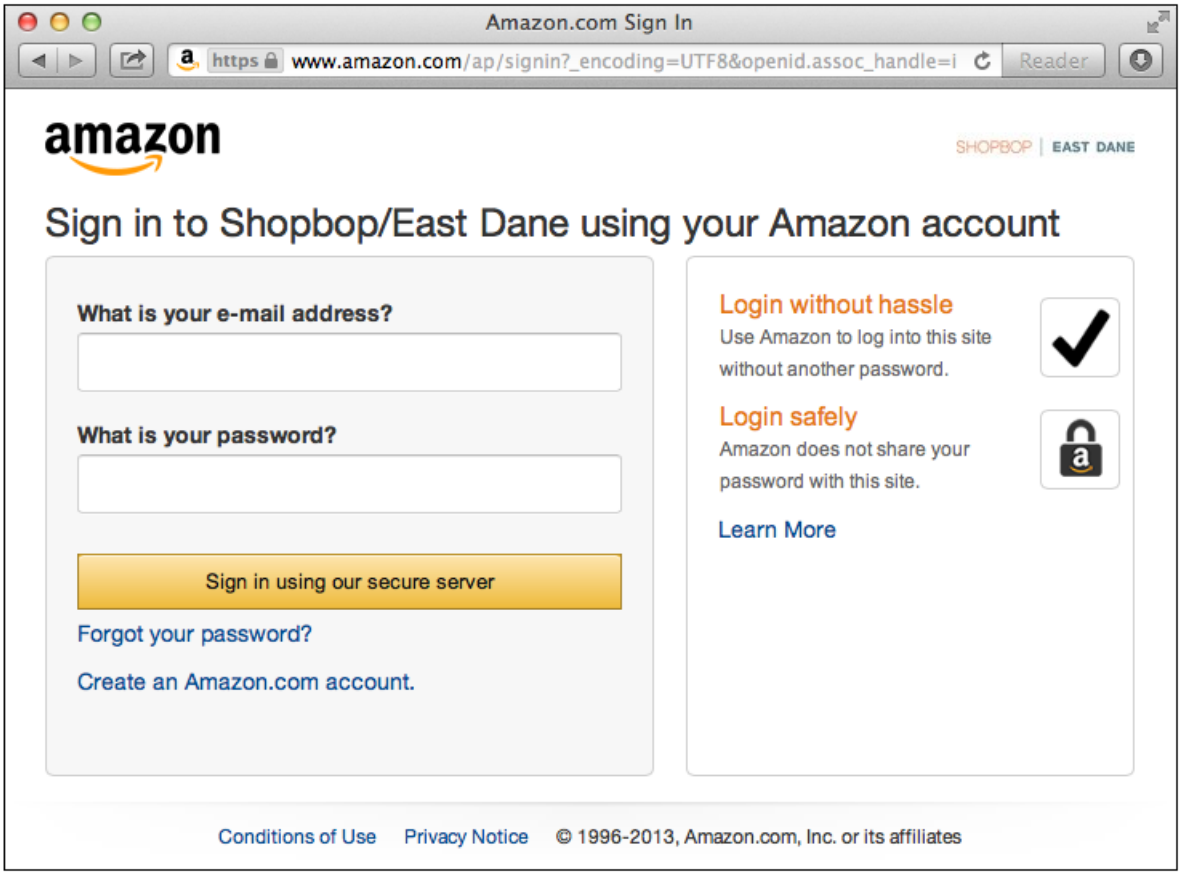


Figure 2: Login with Amazon login screen

If this is the first time users have logged in from this website or app, Amazon presents them with a list of permissions requested by the client. Clients can request the name and email address of the user, and/or request the user's postal (ZIP) code. For example:

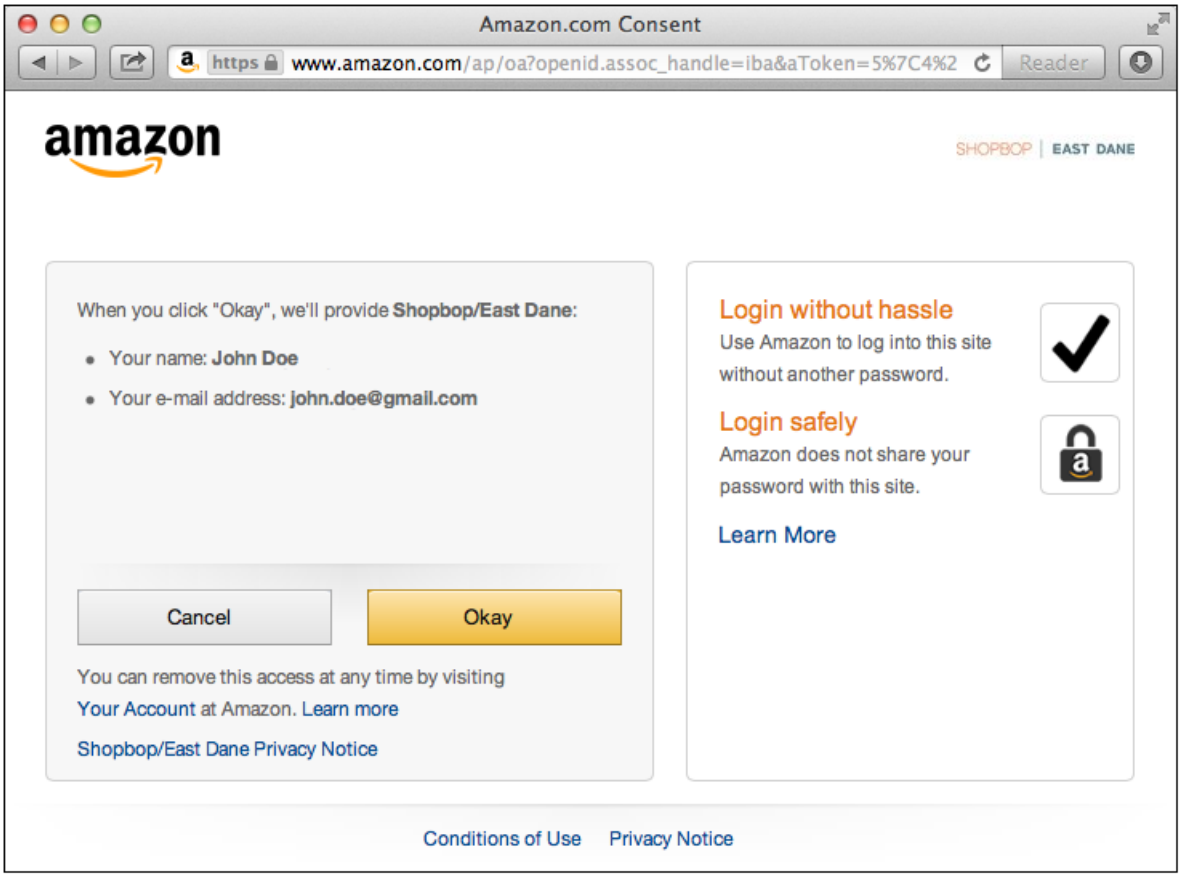


Figure 3: Login with Amazon consent screen

After users log in, the client will use one of the authorization grants to get an access token. The client can then use the access token to access a customer profile, specifying an access scope.

Single Sign-On (SSO) for Web

The login screen for Login with Amazon client websites features a "Keep me signed in" checkbox. When a user checks this box, Login with Amazon will remember the credentials the user supplied as their login (for up to 14 days). When they visit another Login with Amazon website in the same browser, and choose to login, they will get a consent screen or an acknowledgement screen instead of a login screen.

If the website is requesting profile data that requires consent, after a user clicks the Login with Amazon button, they are redirected to a consent screen asking for permission to share their data. When the user clicks Okay, they are redirected back to the Login with Amazon client website.

If the website does not require consent, or a user has already consented, they are presented with an acknowledgement screen after clicking the Login with Amazon button. The acknowledgement screen shows the email address that Login with Amazon remembers, and asks them to confirm that they want to login using that account. When they click **Continue**, they are redirected to the Login with Amazon website.

The acknowledgement screen also gives a user the opportunity to logout of their current account and login again with different credentials. If a user enters a new account and checks the "Keep me signed in" checkbox again, the next time they see an acknowledgement screen it will feature the new account. This does not change their login on previous websites until the website tries to reauthorize the user.

If a user logs out of a website, it does not log them out of other Login with Amazon websites. The single sign-on feature is shared between Login with Amazon third-party websites and the Amazon.com retail site. Amazon.cn,

Amazon.co.jp, and country-specific sites in the European Union have a "Keep me signed in" checkbox but do not participate in single sign-on.

Single Sign-On (SSO) for Mobile

Login with Amazon has implemented support for single sign-on on Kindle Fire, iOS and Android mobile devices.

Under single sign-on, when a user logs into an Amazon-branded app from an iOS or Android device, that login is automatically remembered. If the user opens another Amazon or Login with Amazon mobile app, the app will automatically log them in with the same account, without requiring them to enter any login credentials. On Kindle, Login with Amazon automatically uses the account registered to the device.

When the single-sign on feature is enabled on a device, Login with Amazon apps with an already signed- in account will keep using that account until the user logsout. For more details, please see our Customer Experience Overview for Android/Fire apps, and our Customer Experience Overview for iOS apps.

Access Token

After users log in, they are returned to your website or mobile app. At this point, your client can obtain an *access token* by calling the Login with Amazon authorization service. That token allows clients to access the customer's name and email address from their customer profile.

When you are granted an access token, you may also receive a refresh token. A refresh token is valid for longer than an access token, and allows you to trade in the refresh token for a new access token and a new refresh token.

To access customer data, you must provide an access token to the Login with Amazon authorization service. An access token is an alphanumeric code 350 characters or more in length, with a maximum size of 2048 bytes. Access tokens begin with the characters `Atza|`. Access tokens are only valid for sixty minutes and are specific to the user logging in and the data the app requested when it triggered the login. When you receive an access token, it is as a structure in JSON format with three pieces of information: the `access_token`, the `token_type`, and `expires_in` (the number of seconds before the token expires).These access tokens are bearer tokens, so the `token_type` is always `bearer`. For example:

```
{
   "access_token":"Atza|IQEBLjAsAhRmHjNgHpi0U-Dme37rR6CuUpSR...",
   "token_type":"bearer",
   "expires_in":3600,
   "refresh_token":"AtzrIIQEBLzAtAhRPpMJxdwVz2Nn6f2y-tpJX2DeX..."
}
```

Access tokens are returned in both the Implicit and Authorization Code grants.

An access token is a bearer token and as such can be used by another client. See The OAuth 2.0 Authorization Framework: Bearer Token Usage for more information.

Authorization Code

An *authorization code* is sent to a client as the first step in an Authorization Code Grant. When the client receives the authorization code, it calls the Login with Amazon authorization service with the code, their client identifier and *client secret*. The authorization code is useless by itself, and therefore any malware that intercepts the authorization code cannot impersonate the client to gain an access token.

Refresh Token

A refresh token allows a website to request a new access token, even if the access token has expired. Refresh tokens follow the same format as access tokens, except they begin with the string `Atzr|`. Refresh tokens are valid indefinitely, unless the user has removed the website or mobile app from the list of allowed apps for their account. Refresh tokens have a maximum size of 2048 bytes. A refresh token is specifically assigned to one client and cannot be used by another client.

Refresh tokens are returned only in the Authorization Code Grant.

Customer Profile

A *customer profile* contains the data that Login with Amazon applications can access regarding a particular customer. This includes: a unique ID for the user; the user's name, the user's email address, and their postal code. This data is divided into three *scopes*: `profile`, `profile:user_id` and `postal_code`.

When you request an access token, you can request multiple access scopes by separating them with a space (e.g. `profile postal_code`). When your request is granted, it will specify the scope(s) returned.

profile

The `profile` scope includes a user's name and email address. With access to the customer's profile, you can uniquely identify them when they login, and you can communicate with them via email. The profile data is returned in JSON format and consists of three parts: the `user_id`, the `email`, and the `name`.
The `user_id` is assigned by Amazon, and uniquely identifies the user's account. The `email` is the email address that they have registered with Amazon. Amazon does not verify this email address.

For example:

```
{
    "email" : "johndoe@gmail.com",
    "name" : "John Doe",
    "user_id" : "amznl.account.K2LI23KL2LK2"
 }
```

When a website or app requests access to the `profile` scope, the user will be presented with a consent screen the first time they login. The consent screen shows the information requested and their current values. The user must consent to share this information in order for login to complete. Once the user consents, that consent is recorded and future attempts to login with the same scope will not present a consent screen.

profile:user_id

The second access scope is `profile:user_id`. `profile:user_id` only includes the `user_id` field of the profile. This uniquely identifies the user but does not provide their name, email address, or postal code. Because no personal information is requested, the user will not be presented with a consent screen the first time they log in.

Every company that creates websites or apps for Login with Amazon gets the same `user_id` for a customer. However, when a customer logs in to another company's app or site, the `user_id` will be different. This is so `user_id` cannot be used to track customers across the Web.

postal_code

The third access scope is the `postal_code` scope. This includes the user's zip/postal code number from their primary shipping address. The postal code provides valuable location data that allows you to tune your offerings and understand your customers better. Forexample:

```
{
   "user_id" : "amznl.account.K2LI23KL2LK2"
   "email" : "johndoe@gmail.com",
   "name" : "John Doe",
   "postal_code": "98101",
}
```

When an app requests access to the `postal_code` scope, alone or in concert with the `profile` or `profile:user_id` scope, the user will have to consent to share the information.

Authorization Grants

The Login with Amazon authorization service offers two authorization grants that your website or mobile app can use to authenticate users and access their customer profile. These two grants are the Implicit Grant and the Authorization Code Grant.

The following grant descriptions are in terms of HTTP requests and responses. The mobile SDKs wrap these calls in their own methods and callbacks; however, the overall flow is the same.

Implicit Grant

In the Implicit Grant, a user clicks on a link (or presses a button) (A) that directs them to an Amazon login page. After they login, they are asked to grant an app access to specific profile data (B) and are redirected back to the app. If the user is granted access, an access token is embedded directly in the redirection URI as a URI fragment (C). (This is the implicit grant). The URI fragments, including the access token, are stripped from the redirection URI by the user-agent (the web browser) and the user-agent executes the URI (D). (At this point, the user sees they are logged in to the client and continues using the app normally.) The client website retrieves the access token by using browser-based scripting (e.g. JavaScript) to query the user- agent for the fragments (E). That script can then send the access token to the client (F), or use the access token directly to retrieve the customer profile data from Amazon (G).

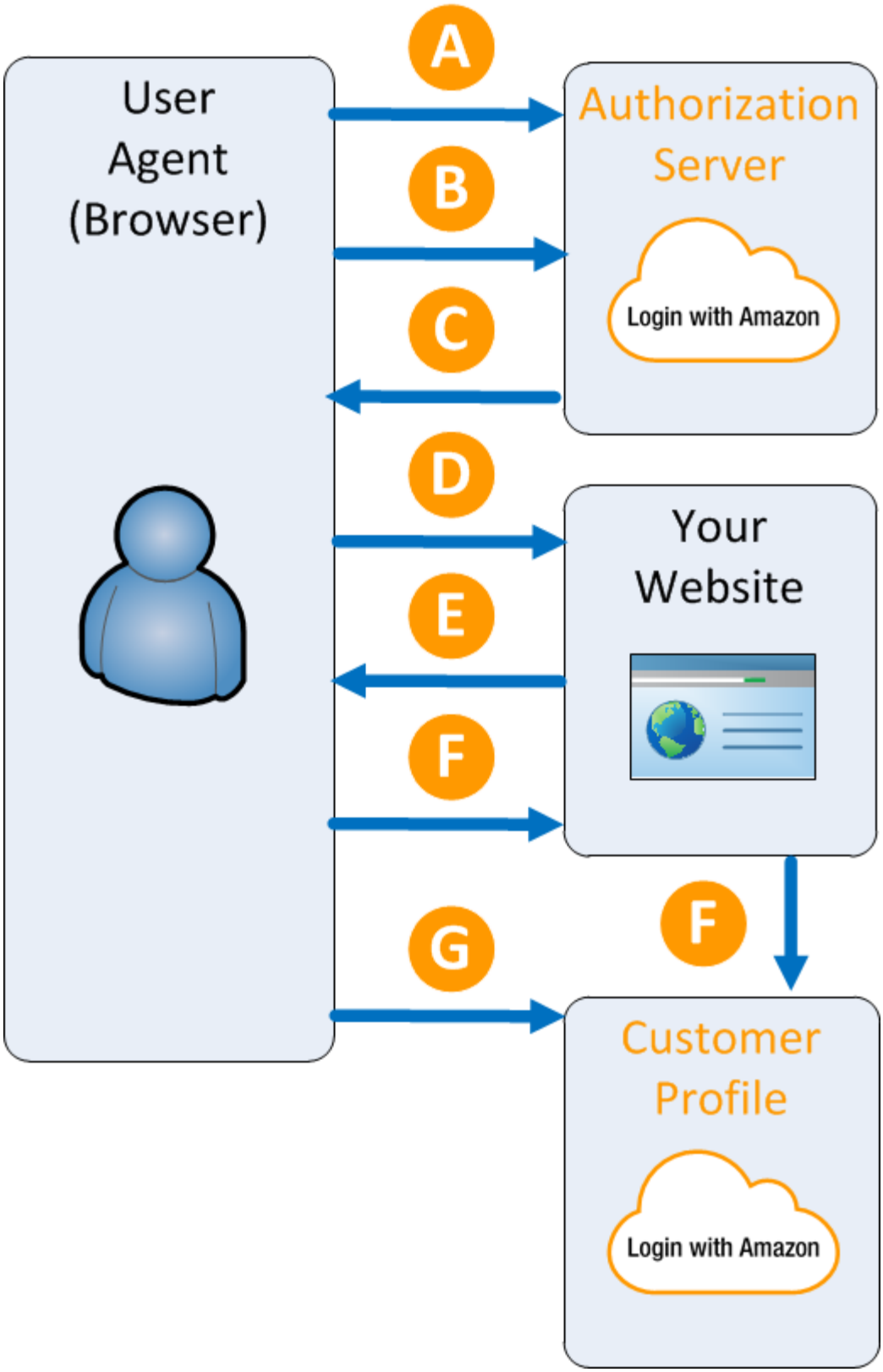


Figure 4: Implicit Grant

Authorization Code Grant

In the Authorization Code Grant, a user clicks on a link (or presses a button) (A) that directs them to an Amazon login page. After they login, they are asked to grant an app access to specific profile data (B) and are redirected back to the app. An authorization code is embedded directly in the redirection URI as a query parameter (C) (this is authorization code grant). The user-agent executes the URI, including the query parameters (at this point, the user sees they are logged in to the app and continues normally). The query parameters are processed directly by the app, and the app uses the authorization code to request an access token directly from the authorization service (D). The authorization code must be paired with a client identifier and client secret, known only to the app. This prevents malicious software from intercepting the authorization code and impersonating the app.

Once the authorization code, client identifier and client secret are verified, the app is granted an access token and a refresh token from the authorization service (E). They can use the access token to access the customer profile data from Amazon. When the access token expires, they can use the refresh token to gain a new access token and a new refresh token.

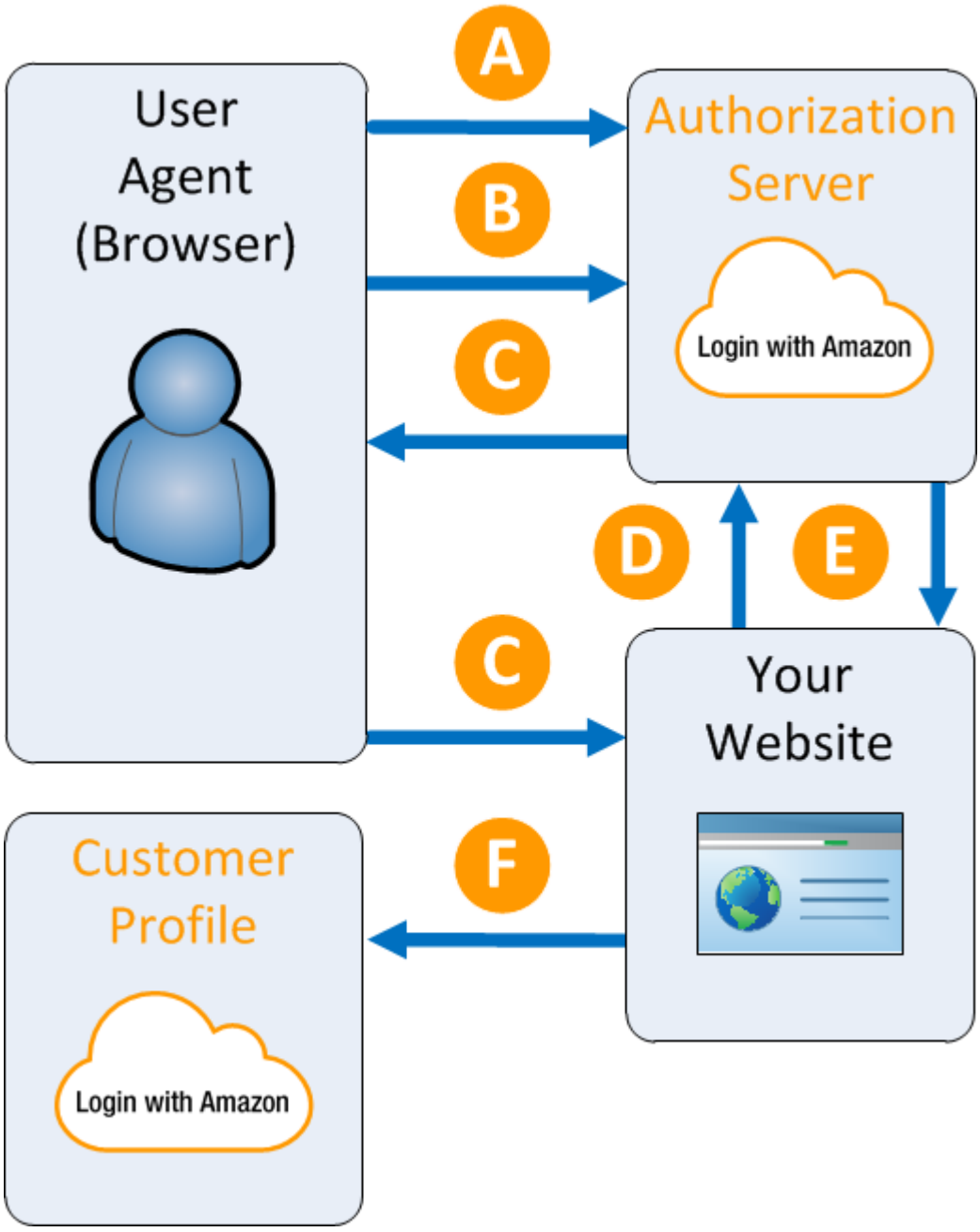


Figure 5: Authorization Code Grant

Client Application

Before you can use Login with Amazon on a website or in a mobile app, you must register an application with Login with Amazon. Your Login with Amazon application is the registration that contains basic information about your business, and information about each website or mobile app you create that supports Login with Amazon. This business information is displayed to users each time they use Login with Amazon on your website or mobile app. Users will see the name of your application, your logo, and a link to your privacy policy. To create an application, clients must supply the following:

- **Name**. This name will be displayed on the consent screen when a user is asked to give your website or mobile app permission to access their profile data. It also appears on the list of active Login with Amazon clients in the Your Account section for Amazon users.
- **Description**. The description helps you differentiate each of your Login with Amazon applications and is not displayed to users.
- **Privacy Notice URL**. The Privacy Notice URL is the location of your company or application's privacy policy (for example: `http://www.example.com/privacy.html`). This link is displayed to users on the consent screen.

Clients may also supply the following:

- **Logo Image File**. This logo is displayed on the sign-in and consent screen when users log into your website or mobile app. The logo will be shrunk to 50 pixels in height if it is taller than 50 pixels; there is no limitation on the width of the logo. The following formats are accepted: PNG, JPEG, and GIF.

After creating an application, you can register each website or mobile app that will use Login with Amazon.

For more information, including registration for individual websites, see Register your Login with Amazon Application.

Client Identifier

When clients create a website or mobile app, they are assigned a *client identifier* and a *client secret*. Client identifiers and client secrets are assigned in pairs. An app can have multiple client identifiers.

The client identifier is used to identify your app, either alone or along with the client secret. Both authorization grants use the client identifier, but the Authorization Code Grant requires the client secret as well.

The client identifier has a maximum size of 100 bytes. The client secret has a maximum size of 64 bytes.

Using Login with Amazon

Topics

Login with Amazon enables Amazon customers to use their trusted Amazon account to log into websites and mobile apps.

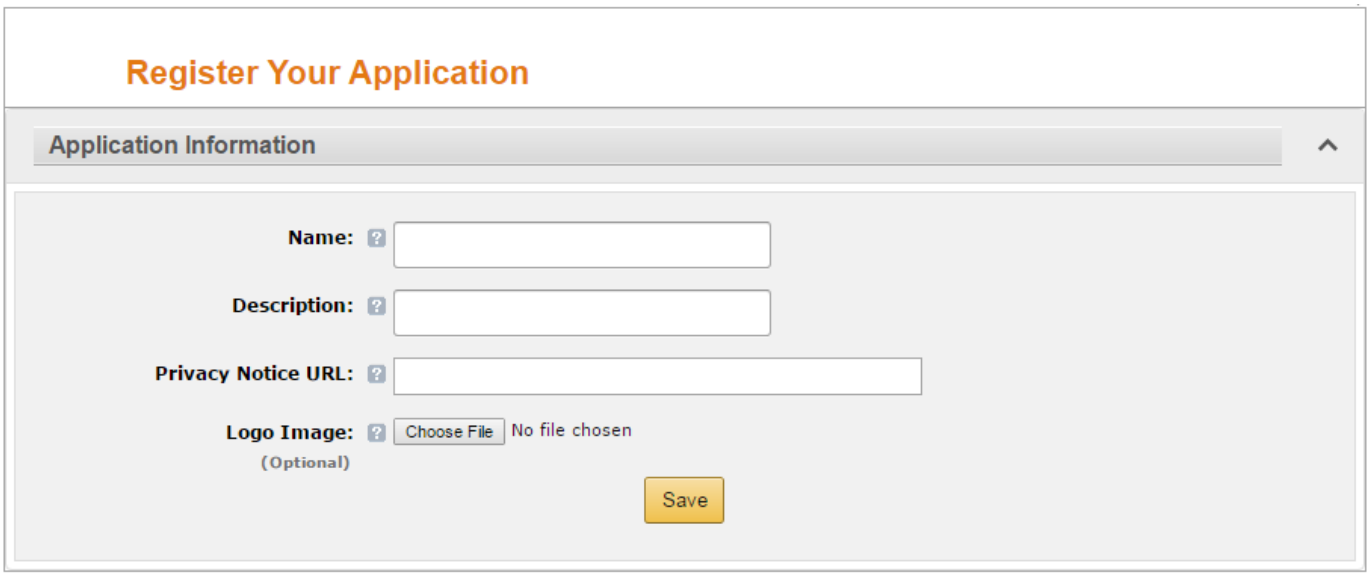
This section discusses how to use Login with Amazon, from creating your application to setting up your website, choosing an authorization grant, and implementing that grant. Once the protocol is implemented, users can login to your site using Login with Amazon and grant you access to their customer profile data.

Register with Login with Amazon

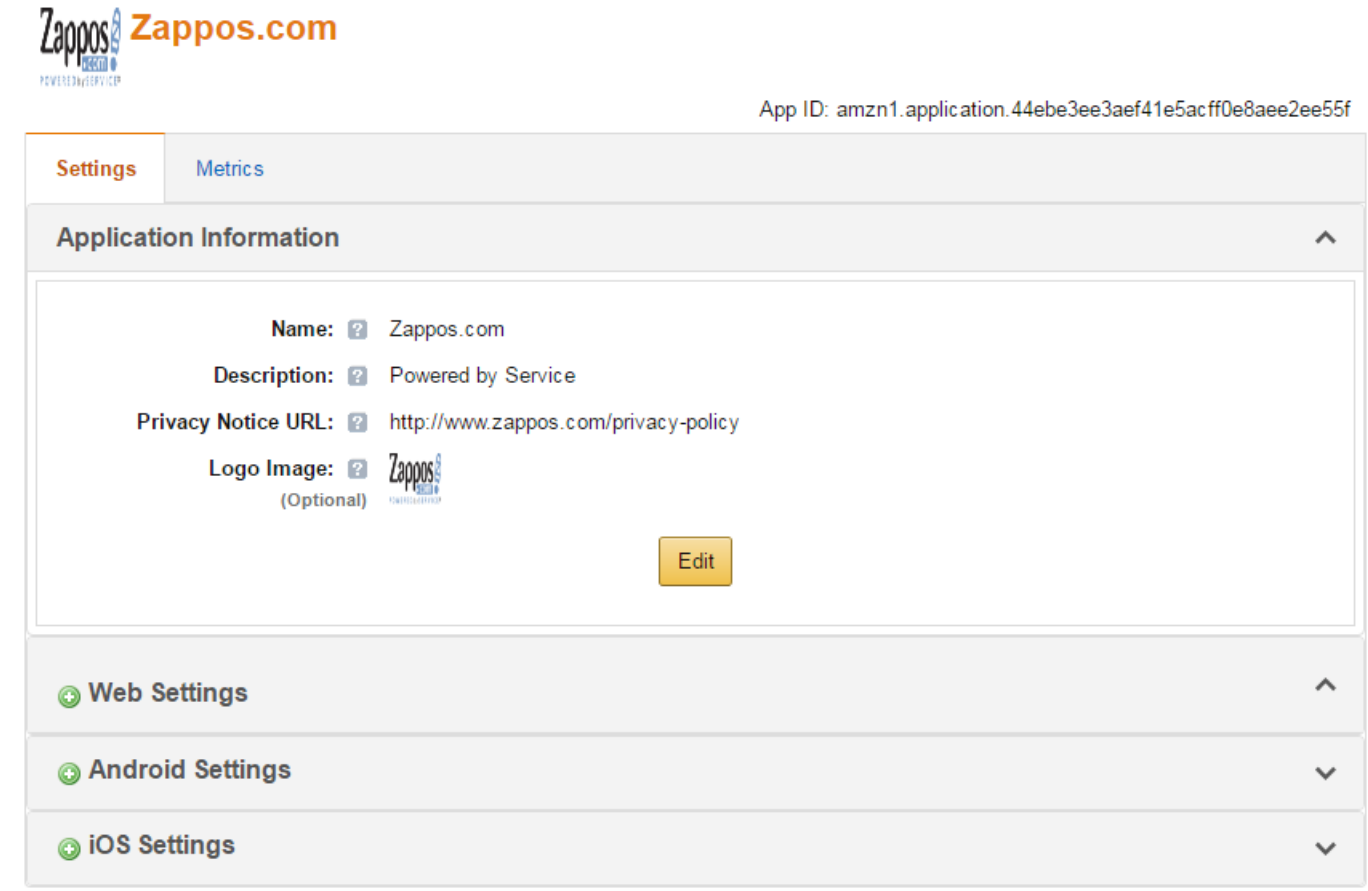
Before you can use Login with Amazon on a website or in a mobile app, you must register an application with Login with Amazon. Your Login with Amazon application contains information about your business, and information about each website or mobile app you create that supports Login with Amazon. The business information is displayed to users each time they use Login with Amazon on your website or mobile app. Users will see the name of your application, your logo, and a link to your privacy policy. These steps demonstrate how to register a Login with Amazon application:

Register Your Login with Amazon Application

1. Go to https://login.amazon.com.
2. If you have signed up for Login with Amazon before, click **App Console**. Otherwise, click **Sign Up**. You will be redirected to Seller Central, which handles application registration for Login with Amazon. If this is your first time using Seller Central, you will be asked to set up a Seller Central account.
3. Click **Register New Application**. The **Register Your Application** form will appear:



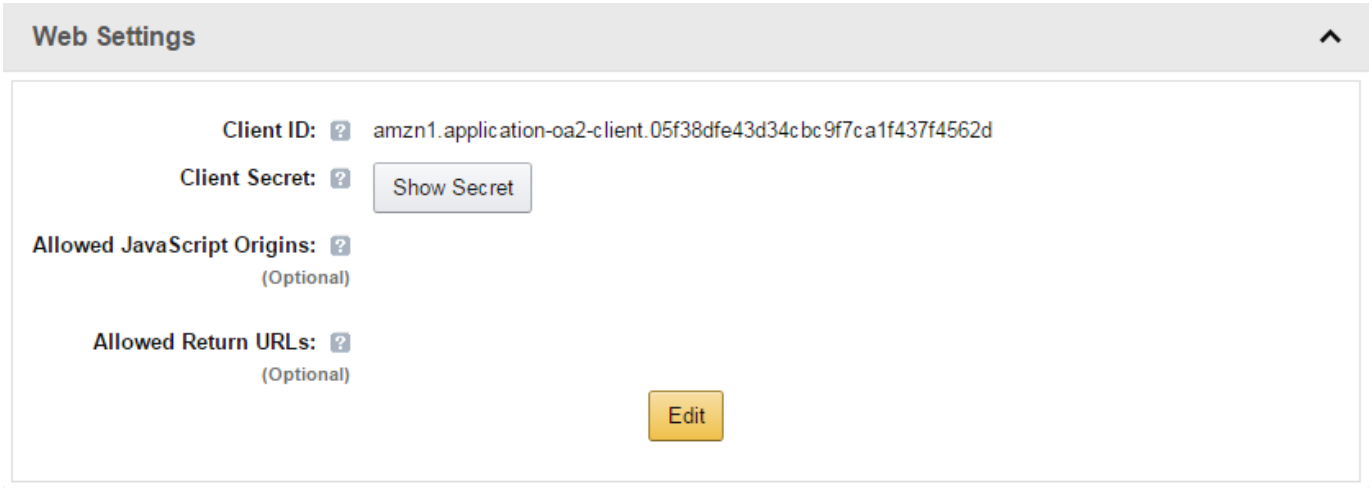
   a. In the Register Your Application form, enter a **Name** and a **Description** for your application.
   The **Name** is the name displayed on the consent screen when users agree to share information with your application. This name applies to Android, iOS, and website versions of your application. The **Description** helps you differentiate each of your Login with Amazon applications and is not displayed to users.
   b. Enter a **Privacy URL** for your application now.
   The **Privacy Notice URL** is the location of your company or application's privacy policy (for example: **http://www.example.com/privacy.html**). This link is displayed to users on the consent screen.
   c. If you want to add a Logo Image for your application, click **Choose File** and locate the applicable image.
   This logo is displayed on the sign-in and consent screen when users log into your website or mobile app. The logo will be shrunk to 50 pixels in height if it is taller than 50 pixels; there is no limitation on the width of the logo. The following formats are accepted: PNG, JPEG, GIF.
4. Click **Save**. Your sample registration should look similar to this:



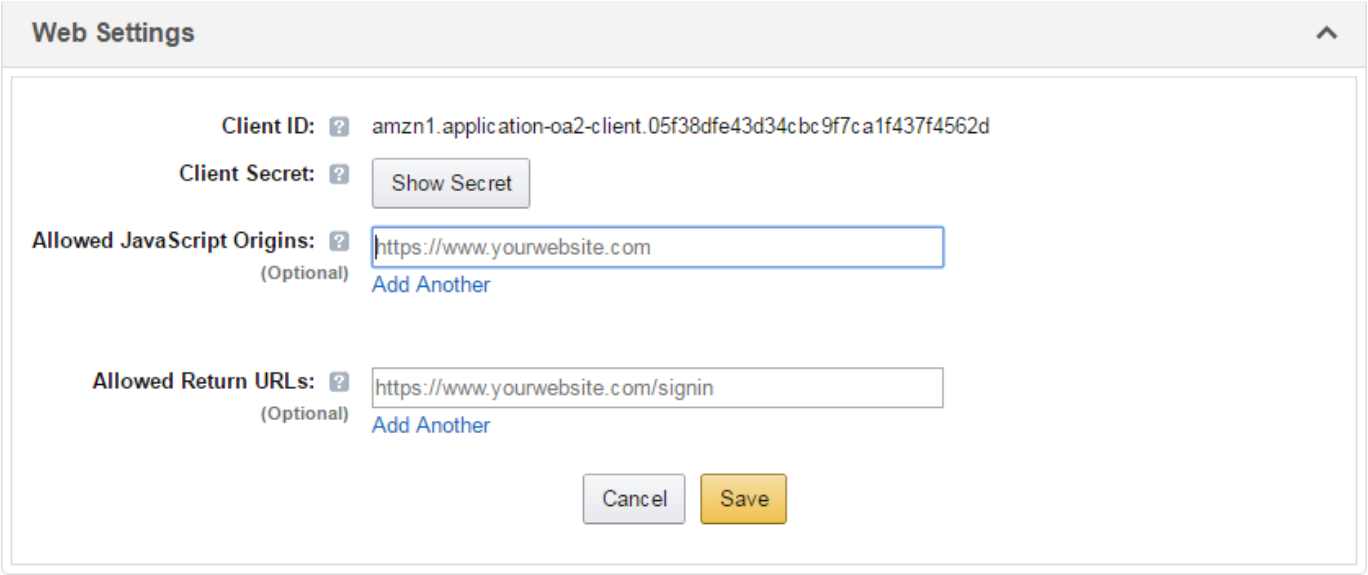
After your basic application settings are saved, you can add settings for specific websites and mobile apps that will use this Login with Amazon account.

Add Web Settings to your Application

1. From the Application screen, click **Web Settings**. You will automatically be assigned values for **Client ID** and **Client Secret**. The client ID identifies your website, and the client secret is used in some circumstances to verify your website is authentic. The client secret, like a password, is confidential. To view the client secret, click **Show Secret**.



2. To add **Allowed JavaScript Origins** or **Allowed Return URLs** to your application, click **Edit**.
   Note: To use Login with Amazon with a website, you must specify either **Allowed JavaScript Origins** or **Allowed Return URLs**. Specify **Allowed JavaScript Origins** to provide a popup authentication experience to your users, or **Allowed Return URLs** to provide a redirect authentication experience. If you are using Amazon Pay, you must specify **Allowed JavaScript Origins**.



   a. If you're providing a popup authentication experience for your users, add your website origin to **Allowed JavaScript Origins**.
      An origin is the combination of protocol, domain name and port (for example: https://www.example.com:8443). Allowed origins must use the HTTPS protocol. If you are using a standard port (port 80 or port 443) you need only include the domain name (for example: https://www.example.com).
      Adding your domain here allows the SDK for JavaScript to communicate with your website directly during the login process. Web browsers normally block cross-origin communication between scripts unless the script specifically allows it.
      To add more than one origin, click **Add another**.
   b. If your website will be making HTTPS calls to the Login with Amazon authorization service and specifying a `redirect_uri` for replies, add those redirect URIs to **Allowed Return URLs**.
      The return URL includes the protocol, domain, path, and query string(s) (for example, https://www.example.com/login.php).
      To add more than one return URL, click **Add another**. If you'd like to dynamically redirect users to different URLs after authentication (for example: if you'd like to redirect each user back to the page they were on prior to logging in) you won't need to list them all here. Instead, initially redirect all users to one static URL (the **Allowed Return URL**). In your authorization request, assign a dynamic value to the `state` parameter and use that value to create a second redirect to the desired URL. For more information, see Dynamically Redirecting Users.
      Note: Redirecting users to a new page for authentication leaves them vulnerable to cross-site request forgery. To secure your app, we strongly recommend using the `state` parameter to validate each authorization response. For more information, see Cross-site Request Forgery.
3. Click **Save**.

Set up your Website

There are a few steps you should take before going live with Login with Amazon on your website. This includes loading the Login with Amazon SDK for JavaScript, adding Login with Amazon graphics, and integrating new Amazon customers into your accounts database.

Load the Login with Amazon SDK for JavaScript

Login with Amazon provides a JavaScript SDK that you may use to obtain access tokens and retrieve customer profiles. Before you can make an access grant call or retrieve a profile, the SDK must load itself from Amazon's content delivery network. To load the SDK, use the following code:

```
<div id "amazon-root"></div>
<script type "text/javascript">
    window.onAmazonLoginReady = function() {
    amazon.Login.setClientId('YOUR-CLIENT-ID');
     };
    (function(d) {
    var a = d.createElement('script'); a.type = 'text/javascript';
    a.async = true; a.id  = 'amazon-login-sdk';
    a.src = 'https://api-cdn.amazon.com/sdk/loginl.js';
    d.getElementById('amazon-root').appendChild(a);
     })(document);
</script>
```

Once the SDK has loaded, it will call `window.onAmazonLoginReady` for initialization. Before using the SDK, you must call `amazon.Login.setClientId`, passing your client identifier. If you do not know your client identifier, visit the App Console at login.amazon.com.

Note: By default, the SDK for JavaScript will display the login screen in a popup window. You can set the `popup` property of the `options` parameter to `false` to instead redirect customers to a new page to login. Popup windows are not supported in native iOS apps. If you intend to use Login with Amazon in your iOS app, we recommend either using the Login with Amazon SDK for iOS, or implementing a redirected login experience. See the Login with Amazon SDK for JavaScript Reference Guide for information on customizing the `options` parameter.

The amazon-root tag

The Login with Amazon SDK for JavaScript requires the `amazon-root` element to be present in the page. The `amazon-root` element must not be hidden using `display: none` or `visibility: hidden`, or some parts of the SDK will not work properly in Internet Explorer.

The SDK inserts elements into `amazon-root` that expect to be positioned relative to the body or relative to an element close to the top of the page. It is best if the `amazon-root` element is not inside an element with `position: absolute` or `position: relative` settings. If you must place the `amazon-root` element inside of a positioned element, you should give it a position close to the top of the body or some parts of the SDK may not work properly.

Add a Login with Amazon Button

Login with Amazon provides standard button graphics for websites and mobile apps. See the Login with Amazon Style Guidelines to see the selection of buttons available and how to use them. Choose the button(s) you will use,

and add them to your website on any page users might like to log in.

Once users have logged in, you should add a "Logout" option (often a hyperlink) to your website. The logout option should delete any cached tokens and remove the user's profile information (such as their name) from the website. Then your website can present the Login button again.

Integrate with your Existing Account System

If your website contains its own user account system, you can take steps to integrate Login with Amazon customers with your existing database. For more information, see Integrate with your existing account system.

Choose an Authorization Grant

The two mechanisms websites can use to obtain access tokens are the Implicit Grant and the Authorization Code Grant. Both authorization grants work by redirecting the user-agent (the user's browser) to Amazon.com for them to login. Once they have logged in, if the website requested an Implicit Grant, the access token is embedded as a fragment in a URI that redirects the user-agent back to the client website. The website then uses a script to obtain the data from the user-agent. If the website requests an authorization code, the user-agent is redirected back to the website and the authorization code is passed as a query string in that URI. The website then makes a secure HTTP call to Amazon behind the scenes to exchange the authorization code for an access token.

Before you implement a Login with Amazon application, you must choose which authorization grant you will use.

Which Grant Type is Right for Your Application?

In general, the advantages of one grant mirror the disadvantages of the other grant.

The advantage of the Authorization Code Grant is that it can be more secure than the Implicit Grant. The user is not involved in the request for the access token, as that takes place directly between the client website and the authorization service. The Authorization Code Grant also features refresh tokens, which gives the client website almost indefinite access to the user's profile data.

The disadvantage to the Authorization Code Grant is that it can be harder to implement, and it relies on server-side scripting. The Authorization Code Grant also uses more round trips than the Implicit Grant.

The advantage of the implicit grant is that it is relatively simple to implement, as it relies on the web browser to receive and store the access token. If the client architecture does not support server-side scripting, this is the only authorization grant that will work with the Login with Amazon authorization service. The Implicit Grant also makes fewer round trips than the Authorization Code Grant.

The disadvantage of the Implicit Grant is that because the user's browser makes the access token request, the user is exposed to the access token. From a strict security perspective, it can be preferable to conceal this information. Also, in the Implicit Grant, when an access token expires, the user must re-authenticate to continue accessing the resources. The Authorization Code Grant features refresh tokens that can be used to obtain a new access token without involving the user.

If you cannot use server-side scripting, the Implicit Grant is your only choice. If you can use server-side scripting, we recommend choosing the Authorization Code Grant.

The Implicit Grant

An Implicit Grant allows a client (typically a website) to direct the user-agent (a user's browser) to a URI at Amazon. The user is then presented with a page asking to grant the website permission to their customer profile. Once the user approves the request, the user-agent is redirected back to the website using a URI that contains an access token in the URI fragment. The user-agent redirects to the client using a redirection URI without the access token fragment, but stores the access token fragment locally. The user- agent then processes a script on the website page that accesses the full redirection URI and passes the fragment information back to the client. For more information, see Implicit Grant.

Authorization Request

To request authorization, the client (website) must redirect the user-agent (browser) to make a secure HTTP call to `https://www.amazon.com/ap/oa` with the following parameters:

Parameter	Description
client_id	REQUIRED. The client identifier. This is provided when you register your website as a client for Login with Amazon. Maximum size of 100 bytes.
scope	REQUIRED. The scope of the request. Must be `profile`, `profile:user_id`, `postal_code`, or some combination, separated by spaces (e.g. `profile%20postal_code`). For more information, see Customer Profile.
response_type	REQUIRED. The type of response requested. Must be token for this scenario.
redirect_uri	REQUIRED. The HTTPS address where the authorization service should redirect the user.
state	RECOMMENDED. An opaque value used by the client to maintain state between this request and the response. The authorization service will include this value when redirecting the user back to the client. It is also used to prevent cross-site request forgery. For more information, see Cross-site Request Forgery.

For example:

```
https://www.amazon.com/ap/oa?client_id=foodev
&scope=profile
&response_type=token
&state=20825757711097519312159189585709344942 4
&redirect_uri=https://client.example.com/auth_popup/token
```

To make an authorization request using the Login with Amazon SDK for JavaScript, you must fill out an `options` object, and call `amazon.Login.authorize`.

```
<script type "text/javascript">

    document.getElementById('LoginWithAmazon').onclick = function() {
      setTimeout(window.doLogin, 1);
      return false;
     };

    window.doLogin = function() {
      options = {};
      options.scope =  'profile';
      amazon.Login.authorize(options, function(response) {
        if ( response.error ) {
          alert('oauth error ' + response.error);
          return;
         }
        amazon.Login.retrieveProfile(response.access_token,
 function(response) {
          alert(response);
         });
       });
     };

</script>
```

The first parameter to `amazon.Login.authorize` is always the `options` object. The second parameter is either a JavaScript function to handle the authorization response, or a redirect URI to another page. The URI must belong to the same domain as the page calling the SDK, and it must be specified using HTTPS.

For example:

```
options = {} ;
options.scope ='profile';
amazon.Login.authorize(options, 'https://mysite.com/redirect_here');
```

Note: If you would like to use the Login with Amazon SDK for JavaScript to request an Implicit grant, you must first have your page load the Login with Amazon SDK for JavaScript. See Load the Login with Amazon SDK for JavaScript.

Once the user has either approved or denied the request, the authorization server will redirect the user to a `redirect_uri`. The client will then receive an Authorization Response.

Authorization Response

After the client (website) directs the user-agent (browser) to make an Authorization Request, the authorization service will redirect the user-agent to a URI specified by the client. If the user granted the request for access, that URI will contain an `access_token` as a URI fragment. For example:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#access_token=Atza|
IQEBLjAsAhRmHjNgHpi0U-Dme37rR6CuUpSR...
&state=2082575771109751931215918958570934494 24
&token_type=bearer
&expires_in=3600
&scope=profile
```

A successful response includes the following values:

Parameter	Description
access_token	The access token for the user account. Maximum size of 2048 bytes.
token_type	The type of token returned. Should be `bearer`.
expires_in	The number of seconds before the access token becomes invalid.
state	The `state` value passed in the authorization request. This value allows you to keep track of the user's state before the request. It is also used to prevent cross-site request forgery. For more information see Cross-site Request Forgery.
scope	The scope of the request. Must be `profile, profile:user_id postal_code`, or some combination.

Note: Some user-agents do not support including a fragment component in the HTTP `Location` response header field. Those clients are not supported.

If you are using the Login with Amazon SDK for JavaScript, the above parameters are available in the `response` object provided by `amazon.Login.authorize`. See Authorization Request for an example.

Once you have obtained an access token, the next step is to use it to read a customer profile. For more details, see Using Access Tokens to Read a Customer Profile.

Authorization Errors

If the user did not grant the request for access, or an error occurs, the authorization service will redirect the user-agent (a user's browser) to a URI specified by the client. That URI will contain error parameters detailing the error. For example:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#error=access_denied
&state=208257577110975193121591895857093449424
```

The error parameters for a failed authorization request include:

Error Parameter	Description
error	An ASCII error code with an error code value.
error_description	A human-readable ASCII string with information about the error; useful for client developers.
error_uri	A URI to a web page with human-readable information about the error; useful for client developers.
state	The client state passed in the original authorization request.

If you are using the Login with Amazon SDK for JavaScript, the above parameters are available in the `response` object provided by `amazon.Login.authorize`. See Authorization Request for an example.

The following error codes can be returned as the value for `error`:

Error Code	Description
invalid_request	The request is missing a required parameter, has an invalid value, or is otherwise improperly formed.
unauthorized_client	The client is not authorized to request an authorization code.
access_denied	The resource owner or authorization server denied this request.
unsupported_response_type	The request specified an unsupported response type. For this scenario, the `response_type` must be `code`.
invalid_scope	The client requested the wrong scope.
server_error	The authorization server encountered an unexpected error. Treat as a 500 Internal Server HTTP error.
temporarily_unavailable	The authorization server is currently unavailable due to a temporary overload or scheduled maintenance. Treat as a 503 Service Unavailable HTTP error.

Verify Access Tokens

Once you receive an access token using the implicit grant, it is highly recommended that you verify the authenticity of the access token before you retrieve a customer profile using that token. If a malicious site can induce a user to login, they can take the valid access token they receive and use it to mimic an authorization response to your site.

To verify a token, make a secure HTTP call to `https://api.amazon.com/auth/O2/tokeninfo`, passing the access token you wish to verify. You can specify the access token as a query parameter. For example:

```
https://api.amazon.com/auth/O2/tokeninfo?access_token=Atza|
IQEBLjAsAhRmHjNgHpi0U-Dme37rR6CuUpSR...
```

Note: Access tokens contain characters that are outside the allowed range for URLs. Therefore, you should URL encode access tokens to prevent errors. For more information, see Section 2.1 of RFC3986.

Token Information Response

If your access token is valid, you will receive the token information as an HTTP response in JSON. For example:

```
HTTP/1.1 200 OK
Date: Fri, 31 May 2013 23:22:10 GMT
x-amzn-RequestId: eb5be423-ca48-11e2-84ad-5775f4514b09
Content-Type: application/json
Content-Length: 247
{
  "iss":"https://www.amazon.com",
  "user_id": "amznl.account.K2LI23KL2LK2",
  "aud": "amznl.oa2-client.ASFWDFBRN",
  "app_id": "amznl.application.436457DFHDH",
  "exp": 3597,
  "iat": 1311280970,
}
```

Compare the `aud` value to the `client_id` you are using for your application. If they are different, the access token was not requested by your application, and you should not use the access token.

A successful response includes the following values:

Parameter	Description
iss	The issuer identifier. This will be https://www.amazon.com.
user_id	The user ID of the account linked to the access token. This is unique for each user and takes the form of `amzn1.account.K2LI23KL2LK2`.
aud	The client identifier used to request the access token. If this does not match the `client_id` used in your authorization request, do not use this token.
app_id	The application identifier of the application that requested the token. The `app_id` is tied to the `client_id` used to request the token, but is not the same value. There can be multiple client identifiers tied to a single application identifier.
exp	The remaining lifetime of the access token, in seconds.
iat	The time the token was issued. The value is number of seconds from `1970-01-01T0:0:0z` as measured in UTC.

If there is a problem verifying the token, you will receive an HTTP error. The error codes for token information include:

Status Code	Error Code	Description
200	success	Success
400	invalid_request	The request is missing a required parameter, has an invalid value, or is otherwise improperly formed.
400	invalid_token	The token provided is invalid or has expired.
500	ServerError	The server encountered a runtime error.

In addition to the error code, you may receive a JSON payload with more information. For example:

```
HTTP/1.1 400 Bad Request
Date: Fri, 31 May 2013 23:21:35 GMT
x-amzn-RequestId: d64bbd14-ca48-11e2-a5dd-ab3bc3c93bae
Content-Type: application/json
Content-Length: 99
 {
   "error": machine-readable error code,
   "error_description": human-readable error description,
 }
```

The Authorization Code Grant

An Authorization Code grant allows a client (typically a website) to direct the user-agent (a user's browser) to a URI at Amazon. The user is then presented with a page asking to grant the website permission to the user's profile. Once the user approves the request, the client receives the authorization code and can trade that code for an access token and refresh token. Once the client has the access token, they can read the customer profile. For more information, see Authorization Code Grant.

If the user refuses the request, the client receives an error from the authorization service.

Authorization Request

To request authorization, the client (website) must redirect the user-agent (browser) to make a secure HTTP call to `https://www.amazon.com/ap/oa` with the following parameters:

Parameter	Description
client_id	REQUIRED. The client identifier. This is provided when you register your website as a client for Login with Amazon. Maximum size of 100 bytes.
scope	REQUIRED. The scope of the request. Must be `profile`, `profile:user_id`, `postal_code`, or some combination, separated by spaces (e.g. `profile%20postal_code`). For more information, see Customer Profile.
response_type	REQUIRED. The type of response requested. Must be `code` for this scenario.
redirect_uri	REQUIRED. The HTTPS address where the authorization service should redirect the user.
state	RECOMMENDED. An opaque value used by the client to maintain state between this request and the response. The authorization service will include this value when redirecting the user back to the client. It is also used to prevent cross-site request forgery. For more information, see Cross-site Request Forgery.

For example:

```
https://www.amazon.com/ap/oa?client_id=foodev
&scope=profile
&response_type=code
&state=20825757711097519312159189585709344942 4
&redirect_uri=https://client.example.com/auth_popup/token
```

To make an authorization request using the Login with Amazon SDK for JavaScript, you must fill out an `options` object, and call `amazon.Login.authorize`.

```
options = {} ;
options.scope = 'profile';
options.response_type='code';
amazon.Login.authorize(options, function(response) {
  if ( response.error )  {
    alert('oauth error ' + response.error);
    return;
   }
  <!-- Pass response.code to your server, and use it to request an
  access token.  The Javascript SDK does not support this step
  because it would expose the client secret.  -->
 });
```

The first parameter to `amazon.Login.authorize` is always the options object. The second parameter is either a JavaScript function to handle the authorization response, or a redirect URI to another page. The URI must belong to the same domain as the page calling the SDK, and it must be specified using HTTPS.

For example:

```
options = {};
options.scope = 'profile';
options.response_type  = 'code';
amazon.Login.authorize(options, 'https://mysite.com/redirect_here');
```

Note: If you would like to use the Login with Amazon SDK for JavaScript to request an Authorization Code Grant, you must first have your page load the Login with Amazon SDK for JavaScript. See Load the Login with Amazon SDK for JavaScript.

Once the user has either approved or denied the request, the authorization server will redirect the user to the `redirect_uri`. The client will then receive an Authorization Response.

Authorization Response

After the client (website) directs the user-agent (browser) to make an Authorization Request, the authorization service will redirect the user-agent to a URI specified by the client. If the user granted the request for access, that URI will contain a `code` parameter containing the authorization code. For example:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBezQQYbYS6WxSbIA
&state=208257577110975193121591895857093449424
```

The authorization code can range from 18 to 128 characters. An authorization code is valid for 5 minutes.

The redirect also copies the `state` passed by the user-agent in the authorization request. This value allows you to keep track of the user's state before the request. It is also used to prevent cross-site request forgery. For more information see Cross-site Request Forgery.

If you are using the Login with Amazon SDK for JavaScript, the above parameters are available in the `response` object provided by `amazon.Login.authorize`. See Authorization Request for an example.

Error responses for this request mirror those used for an implicit grant. See Authorization Errors.

Access Token Request

Once the client (website) receives an Authorization Response with a valid authorization code, it can use that code to obtain an access token. With an access token, the client can read a customer profile (see Access Token). To request an access token, the client makes a secure HTTP POST to `https://api.amazon.com/auth/o2/token` with the following parameters:

Parameter	Description
grant_type	REQUIRED. The type of access grant requested. Must be `authorization_code`.
code	REQUIRED. The code returned by the authorization request.
redirect_uri	REQUIRED. If you provided a `redirect_uri` for the authorization request, you must pass the same `redirect_uri` here. If you used the Login with Amazon SDK for JavaScript for the authorization request, you do not need to pass a `redirect_uri` here.
client_id	REQUIRED. The client identifier. This is set when you register your website as a client.
client_secret	REQUIRED. The secret value assigned to the client during registration.

For example:

```
POST /auth/o2/token HTTP/1.1
Host: api.amazon.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=authorization_code
&code=SplxlOBezQQYbYS6WxSbIA
&client_id=foodev
&client_secret=Y76SDl2F
```

Note: The `client_id` and `client_secret` may be passed in the Authorization header instead, using HTTP Basic authentication. For more information, see RFC2617.

For example:

```
POST /auth/o2/token HTTP/1.1
Host: api.amazon.com
Authorization: Basic czzCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=authorization_code
&code=SplxlOBezQQYbYS6WxSbIA
```

The Login with Amazon SDK for JavaScript does not contain a function for exchanging authorization codes for access tokens. This is because that exchange requires the client secret, which should not be stored in a script. As a result, your web server will need to make the exchange instead.

If you use a `amazon.Login.authorize` to request an authorization code, you should pass the authorization code to your server, or use a `redirect_uri` that will be handled by server-side code.

Access Token Response

When a client (website) makes a secure HTTP POST Authorization Request, the authorization server immediately returns the access token or an error in the HTTP response. For example:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset UTF-8
Cache-Control: no-store
Pragma: no-cache
{
 "access_token":"Atza|IQEBLjAsAhRmHjNgHpi0U-Dme37rR6CuUpSR...",
 "token_type":"bearer",
 "expires_in":3600,
 "refresh_token":"Atzr|IQEBLzAtAhRPpMJxdwVz2Nn6f2y-tpJX2DeX..."
 }
```

A successful response includes the following values:

Parameter	Description
access_token	The access token for the user account. Maximum size of 2048 bytes.
token_type	The type of token returned. Should be `bearer`.

expires_in	The number of seconds before the access token becomes invalid.
Refresh_token	A refresh token that can be used to request a new access token. Maximum size of 2048 bytes.
scope	The scope of the request. Must be `profile, profile:user_id postal_code`, or some combination.

Response parameters are encoded using the `application/json` media type. For more information, see RFC4627.

Access Token Errors

For some errors, the authorization service may return an HTTP 401 (Unauthorized) status code. This includes cases where the client passed the `client_id` and `client_secret` values in the Authorization header and the client could not be authenticated.

An unsuccessful response includes the following values:

Error Parameter	Description
error	An ASCII error code with an error code value.
error_description	A human-readable ASCII string with information about the error; useful for client developers.
error_uri	A URI to a web page with human-readable information about the error; useful for client developers.

The following error codes can be returned as the value for `error`:

Error Code	Description
invalid_request	The request is missing a required parameter, has an invalid value, or is otherwise improperly formed.
invalid_client	The client authentication failed. This isused in cases when the authorization service does not return an HTTP 401 (Unauthorized) status code.
invalid_grant	The authorization code is invalid, expired, revoked, or was issued to a different client_id.
unauthorized_client	The client is not authorized to use authorization codes.
unsupported_grant_type	The client specified the wrong `token_type` in the request.
server_error	The authorization server encountered an unexpected error. Treat as a 500 Internal Server HTTP error.

Use Refresh Tokens

Access tokens will expire after a set time period (normally returned in the `expires_in` parameter). When you obtain an access token, you will also receive a refresh token. You can use a refresh token to retrieve a new access token.

To submit a refresh token, the client makes a secure HTTP POST to `https://api.amazon.com/auth/o2/token` with the following parameters:

Parameter	Description
grant_type	REQUIRED. The type of access grant requested. Must be `refresh_token`.
refresh_token	REQUIRED. The refresh token returned by the original Access Token Response.

For example:

```
POST /auth/o2/token HTTP/1.1
Host: api.amazon.com
Authorization: Basic czzCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=refresh_token
&refresh_token=Atzr|IQEBLzAtAhRPpMJxdwVz2Nn6f2y-tpJX2DeX...
```

Note: The `client_id` and `client_secret` may be passed in the Authorization header instead, using HTTP Basic authentication. For more information, see RFC2617.

For example:

```
POST /auth/o2/token HTTP/1.1
Host: api.amazon.com

Content-Type: application/x-www-form-urlencoded;charset=UTF-8

grant_type=refresh_token
&refresh_token=Atzr|IQEBLzAtAhRPpMJxdwVz2Nn6f2y-tpJX2DeX...
&client_id=foodev
&client_secret=Y76SDl2F
```

The response to a refresh token submission is an Access Token Response.

Dynamically Redirect Users

After users Login with Amazon, they can only be redirected back to the static pages you specified as **Allowed Redirect URLs** when you registered your app. To instead redirect users dynamically to various different URLs after authentication, when you make the authorization request, populate the `state` parameter with a value that can be used to generate the desired redirect URL. For example, if you ultimately want users redirected back to the Item Description page they were viewing prior to authentication, populate the `state` parameter in your request with the unique portion of the Item Description page URL. After authentication, Login with Amazon sends back an authorization response to the client that includes the same state parameter value you specified in the request. The user is sent to the **Allowed Redirect URL**. Use the state parameter value to dynamically generate the URL associated with the page you'd like the user to land on ultimately, then immediately redirect them there from the static page.

If the dynamic URL contains sensitive information, we recommend encrypting, then base64-encoding it, before assigning it to the `state` parameter. When the information is returned in the authorization response, decrypt and decode it to generate the dynamic URL.

In addition, we strongly recommend to anyone using redirect authentication to protect users from cross-site request forgery (CSRF) attacks. Do this by assigning a unique value (a CSRF token) to the `state` parameter in each authentication request, and later validate it in the authentication response. Consider assigning both this

unique csrf token and the redirect URL to the `state` parameter using concatenation.

For example:

```
<csrf-token> + “” + <dynamic-url>
```

For more information on creating a CSRF token, see Cross-site Request Forgery.

Note: This information can be disregarded if your app does not redirect users to a separate page for authentication.

Use Access Tokens to Read a Customer Profile

Once the user grants your website access to their Amazon customer profile, you will receive an access token. To access the authorized customer data, you submit that access token to Login with Amazon using HTTPS. In response, Login with Amazon will return the appropriate customer profile data. The profile data you receive is determined by the `scope` you specified when requesting access. The access token reflects access permission for that scope.

If you are using the Login with Amazon SDK for JavaScript, use `amazon.Login.retrieveProfile` to exchange an access token for a profile. For example:

```
<script type="text/javascript">
  document.getElementById('LoginWithAmazon').onclick = function() {
    setTimeout(window.doLogin, 1);
    return false;
   };
  window.doLogin = function() {
  options = {};
  options.scope = 'profile';
  amazon.Login.authorize(options, function(response) {
    if ( response.error ) {
      alert('oauth error ' + response.error);
    return;
    }
    amazon.Login.retrieveProfile(response.access_token, function(response) {
      alert('Hello, ' + response.profile.Name);
      alert('Your e-mail address is ' + response.profile.PrimaryEmail);
      alert('Your unique ID is ' + response.profile.CustomerId);
      if ( window.console &amp;&amp; window.console.log )
        window.console.log(response);
     });
   });
 };
</script>
```

The `amazon.Login.retrieveProfile` function returns three parameters: `success`, `error`, and `profile`. `success` indicates whether the call was successful. `error` contains an error message if an error occurred. If there was no error, `profile` contains the user's profile.

Note: If you would like to use the Login with Amazon SDK for JavaScript to request a customer profile, you must first have your page load the Login with Amazon SDK for JavaScript. See Load the Login with Amazon SDK for JavaScript.

If you are calling the endpoint directly, you can specify the access token in one of three ways: as a query parameter, as a bearer token, or using `x-amz-access-token` in the HTTP header.

For example:

```
https://api.amazon.com/user/profile?access_token AtzaIIQEBLjAsAhRmHjNgHpi0U-
Dme37rR6CuUpSR...
```

```
GET /user/profile HTTP/1.1
Host: api.amazon.com
Date: Wed, 01 Jun 2011 12:00:00 GMT
Authorization: Bearer Atza|IQEBLjAsAhRmHjNgHpi0U-Dme37rR6CuUpSR...
```

```
GET /user/profile HTTP/1.1
Host: api.amazon.com
Date: Wed, 01 Jun 2011 12:00:00 GMT
x-amz-access-token: Atza|IQEBLjAsAhRmHjNgHpi0U-Dme37rR6CuUpSR...
```

Note: Access tokens contain characters that are outside the allowed range for URLs. Therefore, you should URL encode access tokens to prevent errors. For more information, see Section 2.1 of RFC3986.

Login with Amazon only supports `application/json` as a content type and `en-us` as a content language. Login with Amazon uses this content type and language by default, even if they are not specified.

```
GET /user/profile HTTP/1.1
Host: api.amazon.com
Date: Wed, 01 Jun 2011 12:00:00 GMT
x-amz-access-token: Atza|IQEBLjAsAhRmHjNgHpi0U-Dme37rR6CuUpSR...
Accept: application/json
Accept-Language: en-US
```

Customer Profile Response

If your access token is valid, you will receive the customer's profile data as an HTTP response in JSON. For example:

```
HTTP/1.1 200 OK
x-amzn-RequestId: 0f6bef6d-705c-11e2-aacb-93e6bf269301
Content-Type: application/json
Content-Language: en-US
Content-Length: 85
{
  "user_id": "amzn1.account.K2LI23KL2LK2",
  "email":"mhashimoto-04@plaxo.com",
  "name" :"Mork Hashimoto",
  "postal_code": "98052"
}
```

The `Request-Id` is for logging and can be ignored. If you are troubleshooting an issue with the Login with Amazon team you may be asked to supply the `Request-Id`.

If there is a problem fulfilling your profile request, you will receive an HTTP error. The error codes for an access request include:

Status Code	Error Code	Description
200	success	Success

400	invalid_request	The request is missing a required parameter, has an invalid value, or is otherwise improperly formed.
400	invalid_token	The token provided is invalid or has expired.
401	Insufficient_scope	The access token provided does not have access to the required scope.
500	ServerError	The server encountered a runtime error.

In addition to the error code, you may receive a JSON payload with more information. For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Content-Length: 74
{
   "error": "machine-readable error code",
   "error_description": "human-readable error description",
   "request_id": "bef0c2f8-e292-4196-8c95-8833fbd559df"
}
```

Log Out Users

Your website should provide a way for users to log out once they have logged in. Once they select the logout option, you should delete any access tokens and refresh tokens associated with that user, and remove their profile information from the website. Your website should then present a login option.

If you are using the Login with Amazon SDK for JavaScript, you can call the `amazon.Login.logout` method to delete any cached tokens. For example:

```
<script type "text/javascript">
    document.getElementById('Logout').onclick   function() {
      amazon.Login.logout();
     };
</script>
```

Subsequent calls to `amazon.Login.authorize` will present the login screen by default.

Security Considerations

Topics

The customer information Login with Amazon provides to participating websites is valuable, and precautions must be taken to ensure it stays confidential. The Login with Amazon protocol makes extensive use of HTTPS to protect communications between the user and Amazon, and between your website and Amazon. This section explains any security threats that go beyond using HTTPS, and explains how you can prevent attackers from gaining valuable customer information.

Cross-site Request Forgery

Cross-site Request Forgery happens when an attacker tricks a user into clicking on a malicious link, where the link goes to a site where the user is currently authenticated. Any commands embedded in that malicious link might be executed automatically because the user is already authenticated on the site, so the user does not see a login screen or any other evidence of malicious activity. In the case of Login with Amazon, Cross-site Request Forgery could be used to mimic a client or an authentication server.

Login with Amazon recommends using the `state` parameter to prevent Cross-site Request Forgery. The client should set the value of the `state` parameter when it initiates an authorization request, and save it to the user's secure session. Unlike the `client_id` and `client_secret` values, in order for the `state` parameter to be useful in preventing attacks it should be unique, and non-guessable, for each and every authorization request. The authorization server returns the same `state` when communicating with the client to deliver authorization codes and access tokens. To protect users from attacks, the client must ignore communication if the returned `state` parameter doesn't match the value from the initial call.

Calculate the State Parameter

Clients can calculate the `state` parameter value in any way they choose, however, the value should be secure from forgery. Login with Amazon recommends using a securely-generated random string with at least 256 bits of entropy. To calculate a `state` value using this method, use a random number generator suitable for cryptographic operations.

Here is an example in Python:

```
def generate_state_parameter():
      random = os.urandom(256)
      state = base64.b64encode(random)
      return (state)
```

After generating the `state` parameter value, save it to the user's session information, ensuring the information is communicated securely and saved to a secure session. When the `state` is returned by an authorization response, verify the legitimacy of the user by comparing it with the `state` value saved to their session. If the values do not match, you should ignore the authorization response.

If you're also using the `state` parameter value to dynamically redirect users after authentication, consider concatenating the securely-generated random string with the dynamic URL, separated by a space (e.g. `state = state + " " + dynamicURL`). When the authorization server returns the `state`, parse it and split it into two values based on the space. The second value will contain the dynamic URL needed to direct the user to the appropriate page after authentication.

Resource Owner Impersonation in Implicit Flow

Websites using the Implicit Grant receive an access token from the Login with Amazon authorization service passively through a redirect URL. If an attacker can entice a user into logging in to a malicious site, the attacker's site will receive a legitimate access token. The attacker can then pass that access token to the redirect URL on another site to make it appear that the user is trying to login to the site.

Clients using the implicit flow can guard against this attack by verifying that an access token is legitimate before

using it to retrieve a customer profile and complete login. Login with Amazon provides an endpoint specifically for verifying access tokens. Clients should use that endpoint to compare their client identifier to the client identifier that originally requested the access token. If the client identifiers do not match, the login request should be rejected.

For more information, see Verifying Access Tokens.

Open Redirectors

An open redirector is an endpoint configured to redirect a user-agent based on the value of a parameter, without any kind of validation. Open redirectors can be exploited in Login with Amazon by attackers who fool users into authorizing access to the legitimate website, but when the authorization server redirects to the client, the open redirector sends it back to the attacker.

Login with Amazon client websites should ensure that the target of the redirection URI they use for authentication is not configured as an open redirector.

Some common patterns for open redirectors are:

```
example.com/go.php?url=
example.com/search?q=user+search+keywords&url=
example.com/coupon.jsp?code=ABCDEF&url=
example.com/login?url=
```

Code Injection

A code injection attack happens when an attacker changes the value of an input or a parameter in a way that causes unexpected behavior in a website (such as a Login with Amazon client). A code injection attack is possible when a website does not validate incoming data before acting on it.

Login with Amazon client websites should validate data coming from the authorization service, especially the `state` parameter, before acting on it. Login with Amazon clients should also validate customer profile data if they use it programmatically.

Glossary

access scope	An access scope defines the type of user profile data the client is requesting. The first time a user logs in, they see a list of the items in the access scope and must agree to provide the data to the client in order to proceed.
access token	An access token is granted by the authorization server when a user logs in to a site. An access token is specific to a client, a user, and an access scope. Access tokens have a maximum size of 2048 bytes. A client must use an access token to retrieve customer profile data.
allowed JavaScript origins	A JavaScript origin is the combination of protocol, domain, and port where a JavaScript call originates. By default, web browsers block JavaScript calls from one origin that try to call script on another origin. The Login with Amazon SDK for JavaScript allows calls from other origins if they are specified as part of an application. When registering a website for Login with Amazon, enter the scheme, domain, and optionally the port, of the webpage which includes the Login with Amazon SDK for JavaScript (for example, http://www.example.com or https://localhost:8080).
allowed return URL	A return URL is an address on a website that uses Login with Amazon. The authorization service redirects users to this address when they complete login. See also redirect URL.
API key	This is an identifier that Login with Amazon SDKs use to identify a mobile app to the authorization service. API keys are generated when you register a mobile app.
application	An application is the registration that contains information the authorization service needs to verify a client before that client can access customer profiles. It also contains basic information about your business that is displayed to users each time they use Login with Amazon on your website or mobile app.
appstore ID	An AppStore ID uniquely identifies a mobile app in the Amazon AppStore.
Authorization code	An authorization code is a value used by the Authorization Code grant to allow a website to request an access token.
authorization code grant	An Authorization Code grant is an authorization grant that uses server-based processing to request an access token. Using the authorization code grant, the server receives an authorization code as a query parameter after the user logs in. The server exchanges the authorization code, client identifier, and client secret for an access token and a refresh token.
authorization grant	An authorization grant is the process where the authorization service verifies a client website's request for access to a customer profile. An authorization grant requires a client identifier and an access scope, and may require a client secret. If the process succeeds, the website is granted

	an access token. There are two types of authorization grants, an Implicit grant and an Authorization Code grant.
authorization service	The Login with Amazon authorization service is the collection of end points provided by Amazon that allows a client to login a user through authorization grants. The authorization service presents the login screen and the permissions screen to users. It provides access tokens, refresh tokens, and customer profile data to Login with Amazon clients.
bundle identifier	The bundle identifier is a unique identifier for an iOS app. They normally take the form of `com.companyname.appname`.
client	A client is a website or mobile app that uses Login with Amazon.
client identifier	The client identifier is a value assigned to the client when they register with Login with Amazon. It has a maximum size of 100 bytes. The client identifier is used in conjunction with the client secret to verify the identity of the client when they request an authorization grant from the authorization service. The client identifier is not secret.
client secret	The client secret, like the client identifier, is a value assigned to the client when they register with Login with Amazon. It has a maximum size of 64 bytes. The client secret is used in conjunction with the client identifier to verify the identity of the client when they request an authorization grant from the authorization service. The client secret must be kept confidential.
consent screen	When a user logs into a website or mobile app for the first time, they are presented with a consent screen if the app requests profile data. The consent screen shows the name, logo image file, and privacy notice URL associated with app, along with the access scope the app is requesting.
customer profile	A customer profile contains information about the Login with Amazon customer, including their name, email address, postal code, and a unique identifier. A website must obtain an access token before they can obtain a customer profile. The kind of profile data returned is determined by the access scope.
implicit grant	An Implicit Grant is an authorization grant that can be completed using only the user's web browser. Using the implicit grant, the browser receives an access token as a URI fragment. An implicit grant requires a client identifier and an access scope. The implicit grant does not return a refresh token.
login screen	The login screen is an HTML page presented to users when they try to login to a website or mobile app using Login with Amazon. Users can enter an existing Amazon account or create a new one from this page.
logo image file	A PNG file provided by the client when setting up an application. This is displayed on the permissions screen if the user has not granted access to the client website. The logo represents the client website.

package name	A package name is a unique identifier for an Android app. They normally take the form of `com.companyname.appname`.
privacy notice URL	A URL provided by the client when setting up an application. This is displayed on the consent screen if the user has not granted access to the client website. The URL should direct users to the privacy policy for the client website.
redirect URL	A URL provided by the client to the authorization service. After the user logs in, the service will redirect the user's browser to this address. See also Allowed Return URL.
refresh token	A refresh token is granted by the authorization service when the client uses the Authorization Code grant. A client can use a refresh token to request a new access token when the current access token expires. Refresh tokens have a maximum size of 2048 bytes.
signature	A signature is a SHA-256 hash value embedded in a mobile app that verifies the identity of the app. They normally take the form of `01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef`.
user	A user is a person who visits a client website and tries to log in using Login with Amazon.
version	A version is a particular type of Login with Amazon client registered to an application. A Login with Amazon application can have multiple versions,