

Selling Partner API Developer Guide

Contents

What is Selling Partner API?	4
Selling Partner API HTTP methods	4
Selling Partner API endpoints	4
marketplaceId values	5
Hybrid Selling Partner API applications	6
Global applications	7
Registering your Selling Partner API application	8
Step 1. Create an AWS account	8
Step 2. Create an AWS IAM policy	8
Step 3. Create an IAM user	8
Step 4. Provide your application registration information	9
Viewing your developer information	11
Authorizing Selling Partner API applications	12
Marketplace Appstore workflow	12
Step 1. The seller initiates authorization from the Marketplace Appstore	13
Step 2. The seller consents to authorize your application	13
Step 3. The seller signs into your website	13
Step 4. Amazon sends you the authorization information	14
Step 5. Your application exchanges the LWA authorization code for an LWA refresh token	15
Website workflow	16
Step 0. Set up your OAuth authorization URIs	16
Step 1. The seller initiates authorization from your website	16
Step 2. The seller consents to authorize your application	17
Step 3. Amazon sends you the authorization information	17
Step 4. Your application exchanges the LWA authorization code for a LWA refresh token	18
Self authorization	20
Generating a Java SDK with LWA token exchange and authentication	21
Connecting to Selling Partner API using a generated Java SDK	23
Generating a Java client library	25
Connecting to Selling Partner API	26
Step 1. Request a Login with Amazon access token	26
Step 2. Construct a Selling Partner API URI	27
Step 3. Add headers to the URI	28
Step 4. Create and sign your request	28
Credential scope	29
Authorization header	30
Response format	31

Grantless operations.....	33
Throttling: limits to how often you can submit requests.....	34
Include a User-Agent header in all requests.....	35
Selling Partner API sandbox.....	36
How to make a sandbox call to Selling Partner API.....	36
Selling Partner API sandbox endpoints.....	36
Frequently Asked Questions.....	38
How does Selling Partner API differ from Amazon Marketplace Web Service?.....	38

What is Selling Partner API?

Selling Partner API is a REST-based API that helps Amazon sellers programmatically access their data on listings, orders, payments, reports, and more. Applications using Selling Partner API can increase selling efficiency, reduce labor requirements, and improve response time to customers, helping sellers grow their businesses. Selling Partner API builds on the functionality of Amazon Marketplace Web Service (Amazon MWS), but provides features to improve usability and security for developers and the sellers they work with.

Selling Partner API Beta program

Participation in the Selling Partner API Beta program is by invitation only. You will need the help of an Amazon support engineer for tasks such as [registering your Selling Partner API application](#). You can engage an Amazon support engineer for the Beta program by opening a [support case](#). Please include "Selling Partner API Beta" in the subject line of your support case.

Here are the changes to Selling Partner API since the Alpha program:

- You can set up an OAuth authorization workflow that sellers initiate from the Marketplace Appstore detail page. For more information, see [Marketplace Appstore workflow](#).
- You can set up an OAuth authorization workflow that sellers initiate from your own website. For more information, see [Website workflow](#).
- You can create hybrid applications that make calls to both Selling Partner API and Amazon MWS. For more information, see [Pure and hybrid Selling Partner API applications](#).
- We've added new endpoints for the North America, Europe, and Far East regions. See [Selling Partner API endpoints](#).
- We support a number of new marketplaces. See [marketplaceId values](#).

Pure and hybrid Selling Partner API applications

You can build two types of Selling Partner API applications:

- **A pure Selling Partner API application.** This makes calls only to Selling Partner API.
- **A hybrid Selling Partner API application.** This makes calls to both Selling Partner API and Amazon MWS.

For more information, see [Hybrid Selling Partner API applications](#).

Selling Partner API HTTP methods

Selling Partner API supports these HTTP methods.

HTTP method	Description
GET	Retrieves resource data or a list of resources.
POST	Submits an entity to the specified resource, often causing a change in state or side effects on the server.
PUT	Replaces all current representations of the target resource with the request payload.

Selling Partner API endpoints

Selling Partner API endpoints are associated with a particular AWS Region. The AWS Region is important because it is part of the credential scope, which is required for calculating a signature when calling Selling Partner API. For more information, see [Credential scope](#).

Selling region	Endpoint	AWS Region
North America (Canada, US, Mexico, and Brazil marketplaces)	https://sellingpartnerapi-na.amazon.com	us-east-1
Europe (Spain, UK, France, Germany, Italy, Turkey, U.A.E, and India marketplaces)	https://sellingpartnerapi-eu.amazon.com	eu-west-1
Far East (Singapore, Australia, and Japan marketplaces)	https://sellingpartnerapi-fe.amazon.com	us-west-2

marketplaceId values

The `marketplaceId` identifies the marketplace for a request.

North America

Country	marketplaceId	Country code
Canada	A2EUQ1WTGCTBG2	CA
United States of America	ATVPDKIKX0DER	US
Mexico	A1AM78C64UM0Y8	MX
Brazil	A2Q3Y263D00KWC	BR

Europe

Country	marketplaceId	Country code
Spain	A1RKKUPIHCS9HS	ES
United Kingdom	A1F83G8C2ARO7P	GB
France	A13V1IB3VIYZZH	FR
Germany	A1PA6795UKMFR9	DE
Italy	APJ6JRA9NG5V4	IT
Turkey	A33AVAJ2PDY3EV	TR
United Arab Emirates	A2VIGQ35RCS4UG	AE
India	A21TJRUN4KGV	IN

Far East

Country	marketplaceId	Country code
Singapore	A19VAU5U507RUS	SG
Australia	A39IBJ37TRP1C6	AU
Japan	A1VC38T7YXB528	JP

Hybrid Selling Partner API applications

A hybrid Selling Partner API application is an application that can make calls to both Selling Partner API and Amazon Marketplace Web Service (Amazon MWS). Use a hybrid application when your solution requires functionality from both services. When a seller authorizes your hybrid Selling Partner API application, they are (1) authorizing your Amazon MWS developer ID to make calls to Amazon MWS on their behalf, and (2) authorizing the application to make calls to Selling Partner API on their behalf.

Amazon considers a hybrid application to be a single application. For example, when you publish a hybrid application to the Marketplace Appstore, you manage it as a single application.

Creating a hybrid Selling Partner API application

For the Selling Partner API Beta program you need the help of an Amazon support engineer to create a hybrid Selling Partner API application. You can engage an Amazon support engineer using your existing support case for the Beta program or by opening a new [support case](#). Please include "Selling Partner API Beta" in the subject line of your support case.

To create a hybrid Selling Partner API application

1. Publish your Amazon MWS application to the Marketplace Appstore. For information about publishing your application, see [Marketplace Appstore Listing Guide](#) in the Amazon MWS documentation.
2. Register your Amazon MWS application as a Selling Partner API application. See [Registering your Selling Partner API application](#). While you are providing your registration information, ask the Amazon support engineer to convert your Amazon MWS application into a hybrid Selling Partner API application.

Your Amazon support engineer will contact you to confirm that your application is registered and converted into a draft hybrid application. At this point you can set up and test an OAuth authorization workflow. For more information, see [Authorizing Selling Partner API applications](#).

Global applications

You only need to register as a developer once, in the region and marketplace of your choice, to be able to create a pure Selling Partner API application that can be authorized by a seller from any region or marketplace. You need only one set of developer credentials (your AWS access key ID and AWS secret access key) to make calls to any Selling Partner API endpoint, as long as the endpoint is from the same region as the seller who authorized your application. For information about getting your AWS access key ID and AWS secret access key, see [Step 3. Create an IAM user](#).

If you have a hybrid Selling Partner API applications, your calls to Amazon Marketplace Web Service (Amazon MWS) endpoints have the same restrictions as an Amazon MWS application. That is, when you call an Amazon MWS endpoint, you must use Amazon MWS Access Keys associated with the region that the endpoint comes from.

For more information, see:

- [Pure and hybrid Selling Partner API applications](#)
- [Selling Partner API endpoints](#)

Registering your Selling Partner API application

Register your Selling Partner API application starting at [Step 1. Create an AWS account](#). After you register your Selling Partner API application your Amazon support engineer will give you one or more OAuth authorization URIs. You can use an OAuth authorization URI to set up your OAuth authorization workflow. For more information, see [Authorizing Selling Partner API applications](#).

Step 1. Create an AWS account

You need an AWS account because the Selling Partner API security model uses AWS authentication credentials. If you're not already an AWS customer, you can create a free AWS account. For more information, see [AWS Free Tier](#).

Step 2. Create an AWS IAM policy

You need an Identity and Access Management (IAM) policy to define the permissions for the IAM user that you will create in [Step 3. Create an IAM user](#). The IAM policy will give the IAM user permissions to make calls to Selling Partner API.

To create an IAM policy

1. Sign into the AWS Management Console and open the IAM console at console.aws.amazon.com/iam.
2. In the navigation pane at left, click **Policies**.

If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Click **Get Started**.

3. Click the **Create policy** button.
4. Click the **JSON** tab.
5. Paste the following code into the text box, replacing the existing code, and then click **Review policy**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "execute-api:Invoke",
      "Resource": "arn:aws:execute-api:*:*:*"
    }
  ]
}
```

6. On the **Review policy** page, type a **Name** and a **Description** (optional) for the policy that you are creating. We recommend naming your IAM policy, `SellingPartnerAPI`.
7. Review the policy **Summary** to see the permissions that are granted by your policy, then click the **Create policy** button.

Your new IAM policy appears in the list.

For more information, see [Creating IAM Policies](#) in the AWS documentation.

Step 3. Create an IAM user

You need an IAM user to get AWS access keys for authenticating your calls to Selling Partner API. We recommend creating a new IAM user exclusively for this purpose.

We associate your IAM user with your Selling Partner API application during registration.

To create an IAM user

1. Sign into the AWS Management Console and open the IAM console at console.aws.amazon.com/iam.
2. In the navigation pane at left, click **Users** and then click **Add user** button.
3. Type the user name for the new user. This is the sign-in name for AWS.
4. Select **Programmatic access** and then click the **Next: Permissions** button.
5. On the **Set Permissions** page, click **Attach existing policies directly** button.
6. From the list of policies, select the policy that you created in [Step 2. Create an AWS IAM policy](#).



Tip: Filter on **Customer managed** to make your policy easier to find.

7. Click the **Next: Review** button to see all of the choices you made up to this point. When you are ready to proceed, click the **Create user** button.

The AWS access key ID for your new IAM user is displayed.

8. Click **Show** to view the AWS secret access key. To save the AWS access key, click **Download .csv** and then save the file to a safe location.



Important: This is your only opportunity to view or download your AWS secret access key, which you will need to authenticate your calls to Selling Partner API. Save the AWS access key ID and AWS secret access key in a safe and secure place. **You will not have access to the AWS access key again after this step.** If you lose your AWS secret access key you will need to create a new IAM user with its own new set of keys.

9. Click **Close**. In the **User name** column, click your new IAM user and make a note of the User ARN. You will need it in [Step 4. Provide your application registration information](#).

For more information, see [Creating an IAM User in Your AWS Account](#) in the AWS documentation.

Step 4. Provide your application registration information

You will need the help of an Amazon support engineer to register your Selling Partner API application. You can engage an Amazon support engineer for the Beta program by opening a [support case](#). Include "Selling Partner API Beta" in the subject line of your support case.

Please provide the following information to Amazon using your support case.

- **The name of your application.** If you want to create a hybrid application from an Amazon Marketplace Web Service (Amazon MWS) application that you have published on the Marketplace Appstore, provide the name of your Amazon MWS application. For more information, see [Hybrid Selling Partner API applications](#).
- **The regions in which you expect sellers to sign into Seller Central to authorize your application.** You will receive a region-specific OAuth authorization URI for each region that you provide here. For more information, see [Step 0. Set up your OAuth authorization URIs](#).
- **The ARN for your IAM user.** See [Step 3. Create an IAM user](#).
- **A sign-in URI.** This redirects a user-agent to the sign-in page for your website. The sign-in URI must conform to the OAuth 2.0 framework. For more information, see [Authorization Endpoint](#) in the OAuth 2.0 Authorization Framework.
- **A redirect URI.** This redirects a user-agent back to your application. The redirect URI must conform to the OAuth 2.0 framework. For more information, see [Redirection Endpoint](#) in the OAuth 2.0 Authorization Framework.
- **Your seller ID.** We need a seller ID to associate with your Selling Partner API application so you can sign into Developer Central to get developer information for your application. This includes your IAM ARN and your LWA credentials. If this is for a hybrid Selling Partner API application, provide the seller ID(s) that are associated with the developer ID(s) that you used when you registered your application for listing in the Marketplace Appstore. For more information, see [Viewing your developer information](#).
- **Selling Partner API sections.** Indicate the Selling Partner API sections that your application will use.

- **Amazon Marketplace Web Service (Amazon MWS) API sections.** Indicate the Amazon MWS API sections that your application uses.

After we register your application, an Amazon support engineer will use your support case to provide you with the following:

- Confirmation that we have registered your application.
- An application ID for the application that you registered. Use this to identify the application in Developer Central for which you want to get your LWA credentials and IAM ARN. For more information, see [Viewing your developer information](#).
- Regional OAuth authorization URIs for your application. When your application is ready for testing, you can provide links to these OAuth authorization URIs to sellers who are willing to help you test your authorization workflow. We recommend that you limit testing to a minimal number of sellers, as the OAuth authorization URIs will change when your application is published. You can also use the OAuth authorization URIs for initiating authorization from your own website. For more information, see [Website workflow](#).

You can receive up to three regional OAuth authorization URIs, one for each region from which sellers will authorize your application. For more information about regional URIs, see [Step 0. Set up your OAuth authorization URIs](#).

After receiving this information, you can view your developer information and set up your OAuth authorization workflow. For more information, see [Viewing your developer information](#) and [Authorizing Selling Partner API applications](#).

Viewing your developer information

After [registering your Selling Partner API application](#) you can sign into Developer Central to view your developer information.

To view your developer information

1. Go to Developer Central in the region associated with the seller ID that you provided in [Step 4. Provide your application registration information](#):

- North America: <https://sellercentral.amazon.com/sellingpartner/developerconsole>
- Europe: <https://sellercentral.amazon.co.uk/sellingpartner/developerconsole>
- Far East: <https://sellercentral.amazon.co.jp/sellingpartner/developerconsole>

2. Sign in using the credentials for the seller ID from the previous step.

The **Developer Central** page appears, displaying the IAM ARN associated with your application

3. Click **LWA credentials** to view your LWA client identifier and client secret. You will need these credentials to request an LWA access token. For more information, see [Step 1. Request a Login with Amazon access token](#).

Authorizing Selling Partner API applications

The authorization model for Selling Partner API is based on [Login with Amazon](#), Amazon's implementation of OAuth 2.0. In this model the selling partner authorizes your application by interacting with pages displayed by Amazon and by your website. Actions taken by the selling partner trigger responses by your website or by Amazon. The selling partner's browser is the user-agent that passes parameters between your website and Amazon at each selling partner action. To implement OAuth authorization you must configure your website to (1) accept and process the parameters that Amazon passes to it, and (2) redirect the selling partner's browser and pass parameters to Amazon.

Selling partners can authorize your applications using one of these workflows:

- [Marketplace Appstore workflow](#). An OAuth authorization workflow initiated from the Marketplace Appstore detail page.
- [Website workflow](#). An OAuth authorization workflow initiated from your own website.

Grantless operations

A grantless operation is an operation that you can call without explicit authorization from a selling partner. This authorization model doesn't require you to receive and exchange LWA authorization codes and refresh tokens to get an LWA access token, as you must when calling other Selling Partner API operations. Instead you get your LWA access token with a single call to the LWA authorization server.

Migrating authorization from Amazon Marketplace Web Service

If a selling partner has authorized you to make calls to Amazon Marketplace Web Service on their behalf, you can use the Authorization API to migrate that authorization to a hybrid Selling Partner API application. This eliminates the need to request authorization from the selling partner again. For more information, see the Authorization API User Guide.

Marketplace Appstore workflow

The Marketplace Appstore workflow is an OAuth authorization workflow that is initiated from the Marketplace Appstore detail page. When you publish a Selling Partner API application on the Marketplace Appstore, sellers can authorize your application by clicking an **Authorize Now** button on the detail page.

This topic includes the Marketplace Appstore workflow steps as well as information about testing the workflow.

Testing the Marketplace Appstore workflow

Before creating a production Marketplace Appstore workflow, it's important to create a test workflow that can authorize your application in Draft state. Your test workflow won't be exactly the same as the final production workflow. Still, you can test to make sure that your application can exchange parameters with Amazon and receive authorization information.

Here is how a test workflow differs from a production workflow:

- Instead of starting at the Marketplace Appstore detail page, a test workflow starts with a seller directly navigating to the OAuth authorization URI for your application. You can test with a trusted seller who works with you, or you can test the workflow yourself using your own selling account credentials. The OAuth authorization URI must include the `version=beta` parameter to indicate that the workflow is authorizing an application in Draft state. When the seller navigates to the OAuth authorization URI, the workflow continues at [Step 2. The seller consents to authorize your application](#).



Note: If you have more than one regional OAuth authorization URI, be sure give the seller the OAuth authorization URI that corresponds to the region that they sell in.

- Your application adds the `version=beta` parameter to the Amazon callback URI in [Step 3. The seller signs into your website](#). This results in the workflow authorizing your application in Draft state.

When you have finished testing the workflow, update it so that it no longer adds the `version=beta` parameter to the Amazon callback URI in [Step 3. The seller signs into your website](#). This makes it a production workflow. Now any seller can authorize your published application starting from the detail page of the Marketplace Appstore.

The production workflow begins at [Step 1. The seller initiates authorization from the Marketplace Appstore](#).

Step 1. The seller initiates authorization from the Marketplace Appstore

1. The seller signs into Seller Central and goes to the Marketplace Appstore.
2. The seller goes to the detail page for your application and clicks the **Authorize Now** button.

The consent page for your application appears.

Step 2. The seller consents to authorize your application

1. The seller views the consent page, reviews and accepts the data access requested by your application, and then clicks the **Login to [your application name] now** button to continue. The seller can click the **Cancel** button to exit without authorizing.
2. Amazon loads your sign-in URI (which you provided at application registration) into the browser, adding the following query parameters:

Parameter	Description
<code>amazon_callback_uri</code>	A URI for redirecting the browser to Amazon.
<code>amazon_state</code>	A state value generated by Amazon to guard against cross-site request forgery attacks.
<code>selling_partner_id</code>	The seller ID of the seller who is authorizing your application.



Note: If this is a test workflow (the seller started by navigating to your OAuth authorization URI) Amazon includes the `version=beta` parameter. If this is a production workflow (the seller started from the Marketplace Appstore), Amazon does not include the parameter.

For example:


```
https://d2zyfnnpjylxu.cloudfront.net/index.html?
amazon_callback_uri=https://amazon.com/apps/authorize/
confirm/amzn1.sellerapps.app.2eca283f-9f5a-4d13-
b16c-474EXAMPLE57&amazon_state=amazonstateexample
&selling_partner_id=A3FHEXAMPLEYWS
```


Your website's sign-in page appears.

Step 3. The seller signs into your website

1. The seller signs into your website. If the seller does not yet have an account, they complete your registration process.
2. Your application loads the Amazon callback URI (passed by Amazon in the previous step) into the browser, adding the following parameters:

Parameter	Description
<code>redirect_uri</code>	A URI for redirecting the browser to your application.
<code>amazon_state</code>	The <code>amazon_state</code> value passed by Amazon in the previous step.
<code>state</code>	A state value generated by your application. Your application uses this value to maintain state between this request and the response, helping to guard against cross-site request forgery attacks.

Parameter	Description
	 Important: Because OAuth information is passed via URI query parameters, we highly recommended that you do the following: (1) Ensure that the state token is short-lived and verifiably unique to your user, and (2) Set the <code>Referrer-Policy: no-referrer</code> HTTP header, which prevents leaking sensitive information to websites that your website links to. For more information about cross-site request forgery and calculating a state parameter, see Cross-site Request Forgery in the Login with Amazon documentation.

 **Note:** If you include the `version=beta` parameter, the workflow authorizes an application in Draft state. If you do not include the parameter, the workflow authorizes an application published on the Marketplace Appstore.

For example:

```
https://amazon.com/apps/authorize/confirm/
amzn1.sellerapps.app.2eca283f-9f5a-4d13-b16c-474EXAMPLE57?
redirect_uri=https://d2zyfnnpjylxu.cloudfront.net/
landing.html&amazon_state=amazonstateexample&state=-37131022&version=beta
```

OR

```
https://amazon.com/apps/authorize/confirm/
amzn1.sellerapps.app.2eca283f-9f5a-4d13-b16c-474EXAMPLE57?
redirect_uri=https://d2zyfnnpjylxu.cloudfront.net/
landing.html&amazon_state=amazonstateexample&state=-37131022
```

Step 4. Amazon sends you the authorization information

Seller Central briefly displays a page indicating that Amazon is authorizing you to access the seller's data. While this page is displayed, the following actions take place:

1. Amazon loads your redirect URI into the browser, adding the following query parameters:

Parameter	Description
state	The state value that you passed in the previous step.
selling_partner_id	The seller ID of the seller who is authorizing your application.
mws_auth_token	The MWSAuthToken value that you use when you create a query string for a call to Amazon Marketplace Web Service. The <code>mws_auth_token</code> parameter is only passed when the seller is authorizing a hybrid Selling Partner API application. For more information, see Hybrid Selling Partner API applications .
spapi_oauth_code	The Login with Amazon (LWA) authorization code that you exchange for an LWA refresh token. For more information, see Step 5. Your application exchanges the LWA authorization code for an LWA refresh token .

For example:

```
https://client-example.com?
state=state-example
&mws_auth_token=mwsauthtokenexample
&selling_partner_id=sellingpartneridexample
&spapi_oauth_code=spapioauthcodeexample
```

2. Your application validates the state value.
3. Your application saves the `selling_partner_id`, `mws_auth_token` (if passed), and `spapi_oauth_code` values.

4. Your website's landing page displays.

Step 5. Your application exchanges the LWA authorization code for an LWA refresh token

The Login with Amazon SDK for JavaScript can help you with the exchange of an LWA authorization code for an LWA refresh token. For more information, see the Login with Amazon documentation.

- [Add the Login with Amazon SDK for JavaScript](#)
- [Authorization Code Grant](#)

To exchange an LWA authorization code for an LWA refresh token

1. Your application calls the Login with Amazon (LWA) authorization server (<https://api.amazon.com/auth/o2/token>) to exchange the LWA authorization code for an LWA refresh token. The call must include the following query parameters.

Parameter	Description
grant_type	The type of access grant requested. Must be <i>authorization_code</i> .
code	The LWA authorization code that you received in Step 4. Amazon sends you the authorization information .
redirect_uri	The redirect URI for your application.
client_id	Part of your LWA credentials. To get this value, see Viewing your developer information .
client_secret	Part of your LWA credentials. To get this value, see Viewing your developer information .

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=Sp1xl0examplebYS6WxSbIA
&client_id=foodev&client_secret=Y76SDl2F
```

2. The LWA Authorization Server returns the LWA refresh token. The response is in JSON and includes the following elements.

Parameter	Description
access_token	A token that authorizes your application to take certain actions on behalf of a seller. See Connecting to Selling Partner API .
token_type	The type of token returned. Should be <i>bearer</i> .
expires_in	The number of seconds before the access token becomes invalid.
refresh_token	A long-lived token that can be exchanged for a new access token. See Connecting to Selling Partner API .

```
HTTP/1.1 200 OK
Content-Type: application/json;charset UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "access_token": "Atza|IQEBLjAsAexampleHpi0U-Dme37rR6CuUpSR",
  "token_type": "bearer",
  "expires_in": 3600,
  "refresh_token": "Atzr|IQEBLzAtAexamplewVz2Nn6f2y-tpJX2DeX"
}
```

3. Your application saves the `refresh_token` value.
4. The browser displays a page to the seller that indicates next steps for using your application.

An LWA refresh token is a long-lived token that you exchange for an LWA access token, which must be included in every request to Selling Partner API. Once an access token is issued it is valid for one hour. The same access token can be used for multiple API calls, until it expires. See [Connecting to Selling Partner API](#).

Your application is now authorized to make calls to Selling Partner API on the seller's behalf.

For hybrid Selling Partner API applications

If an MWS auth token was returned in [Step 4. Amazon sends you the authorization information](#), your application is also authorized to make calls to Amazon Marketplace Web Service on the seller's behalf. For more information, see [Hybrid Selling Partner API applications](#).

Website workflow

The Website workflow is an OAuth authorization workflow that is initiated from your own website. Sellers sign into your website and click an "Authorize" button that you configure to initiate authorization. For more information, see [Step 0. Set up your OAuth authorization URIs](#).

This topic includes the Website workflow steps as well as information about testing the workflow.

Testing the Website workflow

Before creating a production Website workflow, it's important to create a test workflow that can authorize your application in Draft state. This lets you test to make sure that your application can exchange parameters with Amazon and receive authorization information.

Here is how a test workflow differs from a production workflow:

- Your application adds the `version=beta` parameter to the OAuth authorization URI in [Step 1. The seller initiates authorization from your website](#). This results in the workflow authorizing your application in Draft state.

When you have finished testing the workflow, update it so that it no longer adds the `version=beta` parameter to the OAuth authorization URI in [Step 1. The seller initiates authorization from your website](#). This makes it a production workflow. Now any seller can authorize your published application starting from your own website.

The production workflow begins at [Step 0. Set up your OAuth authorization URIs](#).

Step 0. Set up your OAuth authorization URIs

Set up an "Authorize" button (or something similar) on your application website that the seller can click to initiate authorization of your application. When the seller clicks the button, your website loads an OAuth authorization URI into the browser and the seller is redirected to a Seller Central sign-in page. For information about getting your OAuth authorization URIs, see [Step 4. Provide your application registration information](#).


Multiple regional OAuth authorization URIs


If you have OAuth authorization URIs for more than one region, be sure to set up your "Authorize" buttons so that sellers are redirected to the Seller Central sign-in page for their own region. For more information about getting multiple regional URIs, see [Step 4. Provide your application registration information](#).

Setting up your "Authorize" button(s) is a one-time task.

Step 1. The seller initiates authorization from your website

1. The seller signs into your website. If the seller does not yet have an account, they complete your registration process.
2. The seller clicks the "Authorize" button that you set up in [Step 0. Set up your OAuth authorization URIs](#). If you have more than one regional "Authorize" button, be sure that the seller is directed to the button that corresponds to the region that they sell in.
3. Your application loads the OAuth authorization URI into the browser, adding the following query parameter:

Parameter	Description
state	<p>A state value generated by your application. Your application uses this value to maintain state between this request and the response, helping to guard against cross-site request forgery attacks.</p> <p> Important: Because OAuth information is passed via URL query parameters, we highly recommended that you do the following: (1) Ensure that the state token is short-lived and verifiably unique to your user, and (2) Set the <code>Referrer-Policy: no-referrer</code> HTTP header, which prevents leaking sensitive information to websites that your website links to. For more information about cross-site request forgery and calculating a state parameter, see Cross-site Request Forgery in the Login with Amazon documentation.</p>

 **Note:** If you include the `version=beta` parameter, the workflow authorizes an application in Draft state. If you do not include the parameter, the workflow authorizes an application published on the Marketplace Appstore.

For example:

```
https://sellercentral.amazon.com/apps/authorize/consent?
application_id=appidexample&state=stateexample&version=beta
```

OR

```
https://sellercentral.amazon.com/apps/authorize/consent?
application_id=appidexample&state=stateexample
```

The seller arrives at the sign-in page of Seller Central.

Step 2. The seller consents to authorize your application

1. The seller signs into Seller Central.

The consent page appears.

2. The seller views the consent page, reviews the data access requested by your application, and then clicks the **Confirm** button to continue. The seller can click the **Cancel** button to exit without authorizing.

Step 3. Amazon sends you the authorization information

Seller Central briefly displays a page indicating that Amazon is authorizing you to access the seller's data. While that page is displayed, the following actions take place:

1. Amazon loads your redirect URI into the browser, adding the following query parameters:

Parameter	Description
state	The state value from Step 1. The seller initiates authorization from your website.
selling_partner_id	The seller ID of the seller who is authorizing your application.
mws_auth_token	The MWSAuthToken value that you use when you create a query string for a call to Amazon Marketplace Web Service. The <code>mws_auth_token</code> parameter is only passed when the seller is authorizing a hybrid Selling Partner API application. For more information, see Hybrid Selling Partner API applications.
spapi_oauth_code	The Login with Amazon (LWA) authorization code that you exchange for an LWA refresh token. For more information, see Step 4. Your application exchanges the LWA authorization code for a LWA refresh token.

For example:

```
https://client-example.com?
state=state-example
&mws_auth_token=mwsauthtokenexample
```

```
&selling_partner_id=sellingpartneridexample
&spapi_oauth_code=spapioauthcodeexample
```

2. Your application validates the state value.
3. Your application saves the `selling_partner_id`, `mws_auth_token` (if passed), and `spapi_oauth_code` values.
4. Your website's landing page displays.

Step 4. Your application exchanges the LWA authorization code for a LWA refresh token

The Login with Amazon SDK for JavaScript can help you with the exchange of an LWA authorization code for an LWA refresh token. For more information, see the Login with Amazon documentation.

- [Add the Login with Amazon SDK for JavaScript](#)
- [Authorization Code Grant](#)

To exchange an LWA authorization code for an LWA refresh token

1. Your application calls the Login with Amazon (LWA) authorization server (<https://api.amazon.com/auth/o2/token>) to exchange the LWA authorization code for an LWA refresh token. The call must include the following query parameters.

Parameter	Description
<code>grant_type</code>	The type of access grant requested. Must be <i>authorization_code</i> .
<code>code</code>	The LWA authorization code that you received in Step 3. Amazon sends you the authorization information .
<code>redirect_uri</code>	The redirect URI for your application.
<code>client_id</code>	Part of your LWA credentials. To get this value, see Viewing your developer information .
<code>client_secret</code>	Part of your LWA credentials. To get this value, see Viewing your developer information .

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=Sp1x10examplebYS6WxSbIA
&client_id=foodev&client_secret=Y76SD12F
```

2. The LWA Authorization Server returns the LWA refresh token. The response is in JSON and includes the following elements.

Parameter	Description
<code>access_token</code>	A token that authorizes your application to take certain actions on behalf of a seller. See Connecting to Selling Partner API .
<code>token_type</code>	The type of token returned. Should be <i>bearer</i> .
<code>expires_in</code>	The number of seconds before the access token becomes invalid.
<code>refresh_token</code>	A long-lived token that can be exchanged for a new access token. See Connecting to Selling Partner API .

```
HTTP/1.1 200 OK
Content-Type: application/json;charset UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "access_token": "Atza|IQEBLjAsAexampleHpi0U-Dme37rR6CuUpSR",
  "token_type": "bearer",
  "expires_in": 3600,
  "refresh_token": "Atzr|IQEBLzAtAexamplewVz2Nn6f2y-tpJX2DeX"
}
```

3. Your application saves the `refresh_token` value.
4. The browser displays a page to the seller that indicates next steps for using your application.

An LWA refresh token is a long-lived token that you exchange for an LWA access token, which must be included in every request to Selling Partner API. Once an access token is issued it is valid for one hour. The same access token can be used for multiple API calls, until it expires. See [Connecting to Selling Partner API](#).

Your application is now authorized to make calls to Selling Partner API on the seller's behalf.

For hybrid Selling Partner API applications

If an MWS auth token was returned in [Step 3. Amazon sends you the authorization information](#), your application is also authorized to make calls to Amazon Marketplace Web Service on the seller's behalf. For more information, see [Hybrid Selling Partner API applications](#).

Self authorization

You can self-authorize your application in Developer Central. Before doing this you must follow the steps in [Registering your Selling Partner API application](#).

To implement the full OAuth authorization workflow, see [Authorizing Selling Partner API applications](#).

To self-authorize your application

1. Go to Developer Central in the region associated with the seller ID that you provided in [Step 4. Provide your application registration information](#):
 - North America: <https://sellercentral.amazon.com/sellingpartner/developerconsole>
 - Europe: <https://sellercentral.amazon.co.uk/sellingpartner/developerconsole>
 - Far East: <https://sellercentral.amazon.co.jp/sellingpartner/developerconsole>
2. Sign in using the credentials for the seller ID from the previous step.

The **Developer Central** page appears.

3. Click **Edit** > **Authorize** next to the application that you want to authorize.

A page appears that contains a **Generate refresh token** button.

4. Click **Generate refresh token**.

The Login with Amazon (LWA) refresh token appears. Your application is now authorized to access your selling account.



Important: Click **Generate refresh token** just once to get your refresh token, and then save it for making calls to Selling Partner API. If you click the **Generate refresh token** multiple times you will get a new refresh token each time, invalidating previous refresh tokens.

The refresh token is a long-lived token that you exchange for a short-lived access token. An access token must be included with every request to Selling Partner API. Once an access token is issued it is valid for one hour. The same access token can be used for multiple API calls, until it expires. For more information, see [Step 1. Request a Login with Amazon access token](#).

Generating a Java SDK with LWA token exchange and authentication

These instructions show you how to generate a Java SDK for the Sellers API using [Swagger Code Generator](#) on a computer running Microsoft Windows. The process is the same for users of other operating systems such as macOS or Linux, with the replacement of Windows-specific semantics (for example, C:\). Although these instructions are for the Sellers API, you can modify the instructions to make SDKs for other APIs in Selling Partner API. See the Selling Partner API GitHub repository for Swagger models for each Selling Partner API section.

With this SDK you can make calls to the Sellers API with the following code already set up for you: Login with Amazon token exchange (exchange a refresh token for an access token) and authentication.


To generate a Java SDK with LWA token exchange and authentication

1. Install [Java 8 or newer](#), [Apache Maven 3.6. or greater](#), and [GNU Wget](#) and make them available in your \$PATH.
2. Sign into github.com/amzn.
3. Search for "amazon-marketplace-api-sdk", and then click the **amazon-marketplace-api-sdk** link that appears.
4. Download the **Sellers API/Sellers.json** file.
5. Download the **clients/sellingpartner-api-aa-java** folder. This folder contains an authorization and authentication library, along with customized templates for the Swagger Code Generator.
6. Open a command prompt window and go to a directory where you want to download the Swagger Code Generator.
7. Download the latest version of the Swagger Code Generator.

For example:

```
wget https://repol.maven.org/maven2/io/swagger/swagger-codegen-cli/2.4.13/swagger-codegen-cli-2.4.13.jar -O swagger-codegen-cli.jar
```

swagger-codegen-cli.jar downloads to the current directory.

 **Note:** You can also download from maven.org by directing your browser here: <https://repol.maven.org/maven2/io/swagger/swagger-codegen-cli/2.4.13/swagger-codegen-cli-2.4.13.jar>

8. Copy **Sellers.json** and **swagger-codegen-cli.jar** into a directory structure that makes sense for you. For this example, we'll copy them to C:\SwaggerToCL.
9. Generate the SDK against the templates in the **clients/sellingpartner-api-aa-java** folder.

For example:

```
java -jar C:\SwaggerToCL\swagger-codegen-cli.jar generate -i C:\SwaggerToCL\Sellers.json -l java -t [path to clients\sellingpartner-api-aa-java directory]\resources/swagger-codegen/templates/ -o C:\SwaggerToCL\Sellers_JavaCL
```

The SDK is copied to C:\SwaggerToCL\Sellers_JavaCL

10. Build the AA Library and add it as a dependency of the SDK:

- a. Navigate to the **clients/sellingpartner-api-aa-java** folder and run `mvn package`. This generates a folder named "target". In this folder is a JAR file named **sellingpartnerapi-aa-java-1.0-jar-with-dependencies.jar** (or something similar) and all of the required dependencies.
- b. Install the JAR file in your local Maven repository.

For example:

```
mvn install:install-file -Dfile=[path to JAR file in "target" folder] -DgroupId=com.amazon.sellingpartnerapi -DartifactId=sellingpartnerapi-aa-java -Dversion=1.0 -Dpackaging=jar
```

You can find the actual `groupId`, `artifactId`, and `version` values near the top of the **pom.xml** file in the **clients/sellingpartner-api-aa-java** folder.

- c. Add a dependency on the AA library in the **pom.xml** of the client library:

For example:

```
<dependency>
  <groupId>com.amazon.sellingpartnerapi</groupId>
  <artifactId>sellingpartnerapi-aa-java</artifactId>
  <version>1.0</version>
</dependency>
```

After you have generated your SDK you can use it to make calls to Selling Partner API. See [Connecting to Selling Partner API using a generated Java SDK](#).

Connecting to Selling Partner API using a generated Java SDK

Before your application can connect to Selling Partner API, you must register it and it must be authorized by a seller. See [Registering your Selling Partner API application](#) and [Authorizing Selling Partner API applications](#).

These instructions show you how to use a generated Java SDK to make calls. The SDK exposes classes for configuring your LWA and AWS credentials and uses these to exchange LWA tokens and sign requests for you. For more information, see [Generating a Java SDK with LWA token exchange and authentication](#).

Step 1. Configure your AWS credentials

Create an instance of `AWSAuthenticationCredentials`, using the following parameters:

Name	Description	Required
<code>accessKeyId</code>	Your AWS access key Id. For more information, see Step 3. Create an IAM user .	Yes
<code>secretKey</code>	Your AWS secret access key. For more information, see Step 3. Create an IAM user .	Yes
<code>region</code>	The AWS region to which you are directing your call. For more information, see Selling Partner API endpoints .	Yes

Example:

```
import com.amazon.SellingPartnerAPIAA.AWSAuthenticationCredentials;

...

AWSAuthenticationCredentials
awsAuthenticationCredentials=AWSAuthenticationCredentials.builder()
    .accessKeyId("myAccessKeyId")
    .secretKey("mySecretId")
    .region("us-east-1")
    .build();
```

Step 2. Configure your LWA credentials

Create an instance of `LWAAuthorizationCredentials`, using the following parameters:

Name	Description	Required
<code>clientId</code>	Your LWA client identifier. For more information, see Viewing your developer information .	Yes
<code>clientSecret</code>	Your LWA client secret. For more information, see Viewing your developer information .	Yes
<code>refreshToken</code>	The LWA refresh token. Get this value when the seller authorizes your application. For more information, see Authorizing Selling Partner API applications .	No. Include <code>refreshToken</code> if the operation that you call in the following step requires seller authorization. All operations that are not grantless operations require seller authorization. If you include <code>refreshToken</code> , do not include <code>withScopes</code> .
<code>withScopes</code>	The scope of the LWA authorization grant. You can specify one or more <code>withScopes</code> values.	No. Include <code>withScopes</code> if the operation that you call in the following step is a grantless

Name	Description	Required
	Values: <ul style="list-style-type: none"> <code>SCOPE_NOTIFICATIONS_API</code>. For the Notifications API. <code>SCOPE_MIGRATION_API</code>. For the Authorization API. 	operation . If you include <code>withScopes</code> , do not include <code>refreshToken</code> .
<code>endpoint</code>	The LWA authentication server URL.	Yes

Example for calling operations that require seller authorization:

```
import com.amazon.SellingPartnerAPIAA.LWAAuthorizationCredentials;

...

LWAAuthorizationCredentials lwaAuthorizationCredentials =
    LWAAuthorizationCredentials.builder()
        .clientId("myClientId")
        .clientSecret("myClientSecret")
        .refreshToken("Aztr|...")
        .endpoint("https://api.amazon.com/auth/o2/token")
        .build();
```

Example for calling grantless operations:

```
import com.amazon.SellingPartnerAPIAA.LWAAuthorizationCredentials;
import static
    com.amazon.SellingPartnerAPIAA.ScopeConstants.SCOPE_NOTIFICATIONS_API;
import static
    com.amazon.SellingPartnerAPIAA.ScopeConstants.SCOPE_MIGRATION_API;

...

LWAAuthorizationCredentials lwaAuthorizationCredentials =
    LWAAuthorizationCredentials.builder()
        .clientId("myClientId")
        .clientSecret("myClientSecret")
        .withScopes(SCOPE_NOTIFICATIONS_API, SCOPE_MIGRATION_API)
        .endpoint("https://api.amazon.com/auth/o2/token")
        .build();
```

Step 3. Create an instance of the Sellers API and call an operation

With your `AWSSAuthenticationCredentials` and `LWAAuthorizationCredentials` instances configured you can create an instance of `SellersApi` and call an operation.

Example:

```
SellersApi sellersApi = new SellersApi.Builder()
    .awsAuthenticationCredentials(awsAuthenticationCredentials)
    .lwaAuthorizationCredentials(lwaAuthorizationCredentials)
    .endpoint("https://sellingpartnerapi-na.amazon.com")
    .build();
sellersApi.getMarketplaceParticipations();
```


Generating a Java client library

These instructions show you how to generate a Java client library for the Sellers API using [Swagger Code Generator](#) on a computer running Microsoft Windows. The process is the same for users of other operating systems such as macOS or Linux, with the replacement of Windows-specific semantics (for example, C:\). Although these instructions are for the Sellers API, you can modify the instructions to make client libraries for other APIs in Selling Partner API. See the Selling Partner API GitHub repository for Swagger models for each Selling Partner API section.

While a generated client library can help you make calls to Selling Partner API, it does not contain code for LWA token exchange and authentication. For that, see [Step 1. Request a Login with Amazon access token](#) and [Step 4. Create and sign your request](#). Or, for an SDK that includes LWA token exchange and authentication, see [Generating a Java SDK with LWA token exchange and authentication](#).

To generate a Java client library

1. Install [Java 8 or newer](#), [Apache Maven 3.6. or greater](#), and [GNU Wget](#) and make them available in your \$PATH.
2. Sign into github.com/amzn.
3. Search for "amazon-marketplace-api-sdk", and then click the **amazon-marketplace-api-sdk** link that appears.
4. Download the **Sellers API/Sellers.json** file.
5. Open a command prompt window and navigate to a directory where you want to download the Swagger Code Generator.
6. Download the latest version of the Swagger Code Generator.

For example:

```
wget https://repo1.maven.org/maven2/io/swagger/swagger-codegen-cli/2.4.13/swagger-codegen-cli-2.4.13.jar -O swagger-codegen-cli.jar
```

swagger-codegen-cli.jar downloads to the current directory.



Note: You can also download from maven.org by directing your browser here: <https://repo1.maven.org/maven2/io/swagger/swagger-codegen-cli/2.4.13/swagger-codegen-cli-2.4.13.jar>

7. Copy **Sellers.json** and **swagger-codegen-cli.jar** into a directory structure that makes sense for you. For this example, we'll copy them to C:\SwaggerToCL.
8. Generate the client Library.

For example:

```
java -jar C:\SwaggerToCL\swagger-codegen-cli.jar generate -i C:\SwaggerToCL\Sellers.json -l java -o C:\SwaggerToCL\Sellers_JavaCL
```

The client library is copied to C:\SwaggerToCL\Sellers_JavaCL.

After you have generated your client library you can use it to help you make calls to Selling Partner API. See [Connecting to Selling Partner API](#).

Connecting to Selling Partner API

Before your application can connect to Selling Partner API, you must register it and it must be authorized by a seller. See [Registering your Selling Partner API application](#) and [Authorizing Selling Partner API applications](#).

These instructions show you the steps for making a call to Selling Partner API. For help with constructing a Selling Partner API URI and adding headers to it, see [Generating a Java client library](#). For a more complete solution that includes code for exchanging LWA tokens and authentication, see [Generating a Java SDK with LWA token exchange and authentication](#).

Step 1. Request a Login with Amazon access token

A Login with Amazon (LWA) access token authorizes your application to take certain actions on behalf of a seller. An LWA access token expires one hour after it is issued, and must be included with every request to Selling Partner API.

To request an LWA access token, make a secure HTTP POST to the LWA authentication server (`https://api.amazon.com/auth/o2/token`) with the following parameters:

Name	Description	Required
grant_type	The type of access grant requested. Values: <ul style="list-style-type: none"> <i>refresh_token</i>. Use this for calling operations that require authorization from a seller. All operations that are not grantless operations require authorization from a seller. When specifying this value, include the <i>refresh_token</i> parameter. <i>client_credentials</i>. Use this for calling grantless operations. When specifying this value, include the <i>scope</i> parameter. 	Yes
refresh_token	The LWA refresh token. Get this value when the seller authorizes your application. For more information, see Authorizing Selling Partner API applications .	No. Include <i>refresh_token</i> for calling operations that require authorization from a seller. If you include <i>refresh_token</i> , do not include <i>scope</i> .
scope	The scope of the LWA authorization grant. Values: <ul style="list-style-type: none"> <i>sellingpartnerapi::notifications</i>. For the Notifications API. <i>sellingpartnerapi::migration</i>. For the Authorization API. 	No. Include <i>scope</i> for calling a grantless operation . If you include <i>scope</i> , do not include <i>refresh_token</i> .
client_id	Get this value when you register your application. See Viewing your developer information .	Yes
client_secret	Get this value when you register your application. See Viewing your developer information .	Yes

Example for calling an operation that requires seller authorization:

```
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=Aztr|...
&client_id=foodev
&client_secret=Y76SD12F
```

Example for calling a grantless operation:

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

grant_type=client_credentials
&scope=sellingpartnerapi::notifications
&client_id=foodev
&client_secret=Y76SD12F
```



Tip: To avoid getting an untrusted certificate authority (CA) error when calling the LWA authorization server, be sure to update your trust store so that your application trusts the LWA authorization server.

Response

A successful response includes the following values.

Name	Description
access_token	The LWA access token. Maximum size: 2048 bytes.
token_type	The type of token returned. Must be <i>bearer</i> .
expires_in	The number of seconds before the LWA access token becomes invalid.
refresh_token	The LWA access token that you submitted in the request. Maximum size: 2048 bytes.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "access_token": "Atza|IQEBLjAsAhRmHjNgHpi0U-Dme37rR6CuUpSREXAMPLE",
  "token_type": "bearer",
  "expires_in": 3600,
  "refresh_token": "Atzr|IQEBLzAtAhRPpMJxdwVz2Nn6f2y-tpJX2DeXEXAMPLE"
}
```

For more information, visit the [Authorization Code Grant](#) page in the Login with Amazon documentation.

Step 2. Construct a Selling Partner API URI

Here are the components of a Selling Partner API URI.

Name	Description	Example
HTTP method	One of the Selling Partner API HTTP methods .	GET
Endpoint	A Selling Partner API endpoint .	https://sellingpartnerapi-na.amazon.com
Path	Selling Partner API section/version. number of the section/resource.	/sellers/v1/marketplaceParticipations
Query string	The query parameters.	?marketplace=ATVPDKIKX0DER

For example:

```
GET https://https://sellingpartnerapi-na.amazon.com/sellers/v1/
marketplaceParticipations?marketplace=ATVPDKIKX0DER
```

Step 3. Add headers to the URI

Add headers to the URI that you constructed in [Step 2. Construct a Selling Partner API URI](#).

Here are the HTTP headers that you include in requests to Selling Partner API:

Request headers

Name	Description
host	The marketplace endpoint. See Selling Partner API HTTP methods .
x-amz-access-token	The LWA access token. See Step 1. Request a Login with Amazon access token .
x-amz-date	The date and time of your request.
user-agent	Your application name and version number, platform, and programming language. These help Amazon diagnose and fix problems you might encounter with the service. See Include a User-Agent header in all requests .

Here is an example of a request to Selling Partner API with URI and headers but no signing information:

```
GET https://sellingpartnerapi-na.amazon.com/sellers/v1/
marketplaceParticipations?marketplace=ATVPDKIKX0DER HTTP/1.1
host: sellingpartnerapi-na.amazon.com
user-agent: My Selling Tool/2.0 (Language=Java/1.8.0.221;
Platform=Windows/10)
x-amz-access-token=Atza|IQEBLjAsAhRmHjNgHpi0U-Dme37rR6CuUpSREXAMPLE
x-amz-date: 20190430T123600Z
```

To sign a request to Selling Partner API, see [Step 4. Create and sign your request](#).

Step 4. Create and sign your request

Selling Partner API uses the AWS [Signature Version 4 Signing Process](#) for authenticating requests. When you send HTTP requests to Selling Partner API, you sign the requests so that Amazon can identify who sent them. You sign requests using your AWS access key, which consists of an access key ID and a secret access key. For information about getting your AWS access key, see [Step 3. Create an IAM user](#).



Note: You need to learn how to sign HTTP requests only when you manually create them. When you use the one of the AWS SDKs to calculate signatures for you, the SDK automatically signs the requests with the AWS access key that you specify when you configure it. When you use an SDK you don't need to learn how to sign requests yourself. Java developers, for example, can use [AWS4Signer.java](#) from the AWS SDK for Java as a model for calculating a signature. You can find SDKs for other languages in the [AWS GitHub repository](#).

To create and sign your request, complete the following:

1. Create a canonical request

Follow the instructions in [Task 1: Create a Canonical Request for Signature Version 4](#) in the AWS documentation, using this guidance:

- See [Step 3. Add headers to the URI](#) for an example of an unsigned request to start with when you create your canonical request.
- Use SHA-256 for the hash algorithm.
- Do not put authentication information in the query parameters. Put it in the `Authorization` header parameter. For information about using the `Authorization` header parameter for the authentication information, see [Authorization header](#).

2. Create a string to sign

Follow the instructions in [Task 2: Create a String to Sign for Signature Version 4](#) in the AWS documentation, using this guidance:

- The algorithm designation value is `AWS4-HMAC-SHA256`
 - To determine the credential scope, see [Credential scope](#).
3. Calculate the signature

Follow the instructions in [Task 3: Calculate the Signature for AWS Signature Version 4](#) in the AWS documentation.



Important: Refer to the [Credential scope](#) section to help you complete this step.

4. Add the signing information

Follow the instructions in [Task 4: Add the Signature to the HTTP Request](#) in the AWS documentation, using this guidance:

- Do not add signing information to the query string. Add it to the `Authorization` header parameter.
- See [Authorization header](#) for details about creating an `Authorization` header parameter.

The following example shows what a request might look like after you've added the signing information to it using the `Authorization` header.

```
GET /sellers/v1/marketplaceParticipations?marketplace=ATVPDKIKX0DER
HTTP/1.1
Authorization: AWS4-HMAC-SHA256Credential=AKIDEXAMPLE/20190430/us-east1/
execute-api/aws4_request, SignedHeaders=host;user-agent;x-amz-access-token,
Signature=5d672d79c15b13162d9279b0855cfba6789a8edb4c82c400e06b5924aEXAMPLE
host: sellingpartnerapi-na.amazon.com
user-agent: My Selling Tool/2.0 (Language=Java/1.8.0.221;
Platform=Windows/10)
x-amz-access-token=Atza|IQEBLjAsAhRmHjNgHpi0U-Dme37rR6CuUpSREXAMPLE
x-amz-date: 20190430T123600Z
```

Credential scope

The credential scope is a component of the "string to sign" that you create when you sign a request to Selling Partner API. See [Create and sign your request](#).

Credential scope is represented by a slash-separated string of dimensions, as shown in the following table:

Dimension	Description	Example
Date	An eight-digit string representing the year (YYYY), month (MM), and day (DD) of the request.	20190430
AWS region	The region you are sending the request to. See Selling Partner API endpoints .	us-east-1
Service	The service you are requesting. You can find this value in the endpoint. See Selling Partner API endpoints .	execute-api
Termination string	A special termination string. For AWS Signature Version 4, the value is <code>aws4_request</code>	aws4_request

For example:

```
20190430/us-east-1/execute-api/aws4_request
```



Important: The date that you use as part of your credential scope must match the date of your request, as specified in the `x-amz-date` header. For more information, see [Handling Dates in Signature Version 4](#) in the AWS documentation.

For more information, see [Step 4. Create and sign your request](#).

Authorization header

The `Authorization` header contains the signing information for a request. Although the header is named "Authorization", the signing information is used for authentication.

Here are the components of an Authorization header:

Component	Description
The algorithm used for signing	The hash algorithm used throughout the signing process. Selling Partner API requires SHA-256. You specify this in Step 4. Create and sign your request .
Credential	Your AWS access key ID plus the Credential scope . You get your AWS access key ID in Step 3. Create an IAM user .
SignedHeaders	A list of all the HTTP headers that you included with the signed request. For an example, see Step 3. Add headers to the URI .
Signature	The signature calculated in Step 4. Create and sign your request .

For example:

```
Authorization: AWS4-HMAC-SHA256 Credential=AKIDEXAMPLE/20190430/us-east1/execute-api/aws4_request, SignedHeaders=host;user-agent;x-amz-accesstoken;x-amz-date, Signature=5d672d79c15b13162d9279b0855cfba6789a8edb4c82c400e06b5924aEXAMPLE
```

For more information, see [Step 4. Create and sign your request](#).

Response format

In response to an HTTP request, Selling Partner API returns response headers and a JSON response message.

Response headers

Name	Description
Content-Length	Standard HTTP response header.
Content-Type	Standard HTTP response header.
Date	Standard HTTP response header.
x-amzn-RequestId	Request identifier. Include this if you contact us for support.

Success Response

If your request is successful, Selling Partner API returns the data requested. Here is an example of a successful response:

```
HTTP/1.1 200 OK
Content-Length: 368
Content-Type: application/json
Date: Thu, 0Jun 20122:23:31 GMT
x-amzn-RequestId: 6875f61f-6aa1-11e8-98c6-9b9a3a7283a4
{
  "payload": [
    {
      "marketplace": {
        "id": "ATVPDKIKX0DER",
        "countryCode": "US",
        "name": "Amazon.com",
        "defaultCurrencyCode": "USD",
        "defaultLanguageCode": "en_US",
        "domainName": "www.amazon.com"
      },
      "participation": {
        "isParticipating": true,
        "hasSuspendedListings": false
      }
    }
  ]
}
```

Error response

If your request is unsuccessful, Selling Partner API returns an error response. Here are the elements of the response message in an error response:

Response message

Element	Description	Required
code	HTTP status code.	Yes
message	Explanation of the error condition.	Yes
details	Link to additional information.	No

Here is an example of an error response:

```
HTTP/1.1 400 Bad Request
Content-Length: 117
Content-Type: application/json
Date: Fri, 0Jun 20100:48:2GMT
x-amzn-ErrorType: ValidationException
x-amzn-RequestId: a8c8d99a-6ab5-11e8-b0f8-19363980175b
{
  "errors": [
    {
      "message": "Access to requested resource is denied.",
      "code": "Unauthorized",
      "details": "Access token is missing in the request header."
    }
  ]
}
```


Grantless operations

A grantless operation is an operation that you can call without explicit authorization from a selling partner. This means that when you [request a Login with Amazon access token](#) prior to calling a grantless operation, you don't need to provide a refresh token. Instead, you use the **scope** parameter to provide the scope of the LWA authorization grant. If you use a generated Java SDK (see [Connecting to Selling Partner API using a generated Java SDK](#)) to call grantless operations, use the **withScopes** parameter to set one or more scopes for the LWA authorization grant when you configure your LWA credentials.

See the following table for the grantless operations in the Selling Partner API.

Grantless operations

Operation name	HTTP method and path
createDestination	POST /notifications/v1/destinations
deleteDestination	DELETE /notifications/v1/destinations/{destinationId}
deleteSubscriptionById	DELETE /notifications/v2/subscriptions/{notificationType}/{subscriptionId}
getDestination	GET /notifications/v1/destinations/{destinationId}
getDestinations	GET /notifications/v1/destinations
getSubscriptionById	GET /notifications/v1/subscriptions/{notificationType}/{subscriptionId}
getAuthorizationCode	GET /authorization/v1/authorizationCode

Throttling: limits to how often you can submit requests

Throttling is the process of limiting the number of requests that your application can make to operations in Selling Partner API over a given period of time. Throttling protects the web service from being overwhelmed with requests and helps to ensure that all authorized applications have access to the web service.

Throttling limits for Selling Partner API are applied to every seller-application pair, per operation. This means that the throttling of one operation for a seller-application pair will not affect the throttling of any other operation for that same seller-application pair.

Selling Partner API throttles requests using the [token bucket algorithm](#), where a token counts for a request. The service sets a limit on a steady-state rate and a burst of request submissions against each operation. In the token bucket algorithm, the burst is the maximum bucket size.

Throttling limits for Selling Partner API

Operation	Rate (requests per minute)	Burst
GET/authorization/v1/authorizationCode	60	5
GET/sales/v1/orderMetrics	.5	15
GET/sellers/v1/marketplaceParticipations	.016	15

Include a User-Agent header in all requests

A User-Agent header identifies your application, its version number, and the platform and programming language that you are using. You must include a User-Agent header with every request that you submit to Selling Partner API. Doing this helps Amazon to more effectively diagnose and fix problems, helping to improve your experience using the service.

To create a User-Agent header, begin with the name of your application, followed by a forward slash, followed by the version of the application, followed by a space, an opening parenthesis, the Language name/value pair, and a closing parenthesis. The Language parameter is a required attribute, but you can add additional attributes separated by semicolons.

The following pseudocode illustrates a minimally acceptable User-Agent header:

```
AppId/AppVersionId (Language=LanguageNameAndOptionallyVersion)
```

The following is an example of a User-Agent header that might be used by an application integrator:

```
My Selling Tool/2.0 (Language=Java/1.8.0.221; Platform=Windows/10)
```

If you are a large seller who is integrating through your own IT department, consider creating a User-Agent header that contains the Host attribute, as in the following example. This can help an Amazon support engineer troubleshoot problems for you more effectively.

```
MyCompanyName/build1611 (Language=Perl; Host=jane.desktop.example.com)
```


To specify additional attributes, use the format `AttributeName=Value;`, separating each name/value pair with a semicolon. If you need to use a backslash (`\`), quote it with another backslash (`\\`). Similarly, quote a forward slash in the application name (`\/`), an opening parenthesis in the application version (`\(`), an equal sign in the attribute name (`\=`), and both a closing parenthesis (`\)`), and a semicolon (`\;`) in attribute values.

Because the User-Agent header is transmitted in every request, it is a good practice to limit the size of the header. Selling Partner API will reject a User-Agent header if it is longer than 500 characters.

Selling Partner API sandbox

Selling Partner API provides a sandbox environment that allows you to test your applications without affecting production data or triggering real-world events. Making sandbox calls to Selling Partner API is identical to making production calls except you direct the calls to the [Selling Partner API sandbox endpoints](#). Calling the sandbox endpoints returns static, mocked responses for all Selling Partner APIs. You can refer to these mocked responses in the Swagger model JSON file for the API that you want to call. For more information, see [How to make a sandbox call to Selling Partner API](#).

The Selling Partner API sandbox works like many mocking frameworks, in that it uses pattern matching to return a specified response when the specified parameters are present. A developer receives a response defined in the `x-amazon-spds-sandbox-behaviors` object when they send a request that matches the specified parameters. If the API requires any parameters that aren't specified in the `x-amazon-spds-sandbox-behaviors` object, the sandbox provides the response regardless of the parameter values in the request, as long as the request is valid.

 **Important:** The sandbox is for testing functionality, not scalability testing. Calls to sandbox endpoints are subject to these throttling limits: **rate** = five requests per second; **burst** = 15. For more information about throttling see [Throttling: limits to how often you can submit requests](#).

For the Public Release version of Selling Partner API we will include input validation for sandbox calls so you receive realistic error messages for invalid requests.

How to make a sandbox call to Selling Partner API

Step 1. Get the Swagger model JSON file for the API that you want to call

1. Go to github.com/amzn.
2. Search for "amazon-marketplace-api-sdk", and then click the **amazon-marketplace-api-sdk** link that appears.
3. Download the JSON file for the API that you want to call.

Step 2. Check the JSON file for request parameters

1. Open the JSON file that you downloaded in the previous step.
2. Search for "x-amazon-spds-sandbox-behaviors".

The `x-amazon-spds-sandbox-behaviors` object of the JSON file contains request and response examples for sandbox calls to the API. If the request example contains parameters, use them in the following step.

Step 3. Make a sandbox call to an API

Make a sandbox call to an API in the same way you would make a production call, with these differences:

1. If the request object in the `x-amazon-spds-sandbox-behaviors` object of the JSON file contains one or more parameter/value pairs, specify these in your call.
2. Direct your call to one of the [selling Partner API sandbox endpoints](#).

You should receive a response that matches the payload object contained in the `x-amazon-spds-sandbox-behaviors` object of the JSON file.

Selling Partner API sandbox endpoints

Selling Partner API has sandbox endpoints for the North America, Europe, and Far East selling regions. For more information, see [Selling Partner API sandbox](#).

Selling region	Endpoint	AWS Region
North America (Canada, US, Mexico, and Brazil marketplaces)	https://sandbox.sellingpartnerapi-na.amazon.com	us-east-1
Europe (Spain, UK, France, Germany, Italy, Turkey, U.A.E, and India marketplaces)	https://sandbox.sellingpartnerapi-eu.amazon.com	eu-west-1
Far East (Singapore, Australia, and Japan marketplaces)	https://sandbox.sellingpartnerapi-fe.amazon.com	us-west-2

Frequently Asked Questions

How does Selling Partner API differ from Amazon Marketplace Web Service?

Although Selling Partner API and Amazon Marketplace Web Service (Amazon MWS) are both web services that enable programmatic access to seller data, there are significant differences. Here are some key differences between Selling Partner API and Amazon MWS:

- Selling Partner API treats data as REST-compliant resources that can be accessed and modified via standard HTTP methods. By contrast, Amazon MWS exposes data using operations that are specific to Amazon MWS.
- Selling Partner API authorization leverages LWA, Amazon's implementation of OAuth 2.0. This model eliminates the need for the manual exchange of auth tokens, as required by Amazon MWS. See [Authorizing your Selling Partner API application](#).
- With Amazon MWS, sellers authorize developers. With Selling Partner API, sellers authorize applications. Using Selling Partner API, developers can create multiple applications that require varying levels of access to seller data.
- Selling Partner API provides finer grain data access control than Amazon MWS. Developers can request access to only the data they need, and sellers can grant permissions at the API section, operation, or data resource level.
- Selling Partner API lets you directly procure and manage your own authentication credentials using AWS Identity and Access Management (IAM). With Amazon MWS, you receive authentication credentials provided by Amazon using a special registration workflow, and you get new credentials by opening a contact with Amazon MWS support. See [Step 3. Create an IAM user](#).
- Selling Partner API uses AWS Signature Version 4 for authentication. Amazon MWS uses Signature Version 2. See [Step 4. Create and sign your request](#).