# BOPIS Store Inventory Management

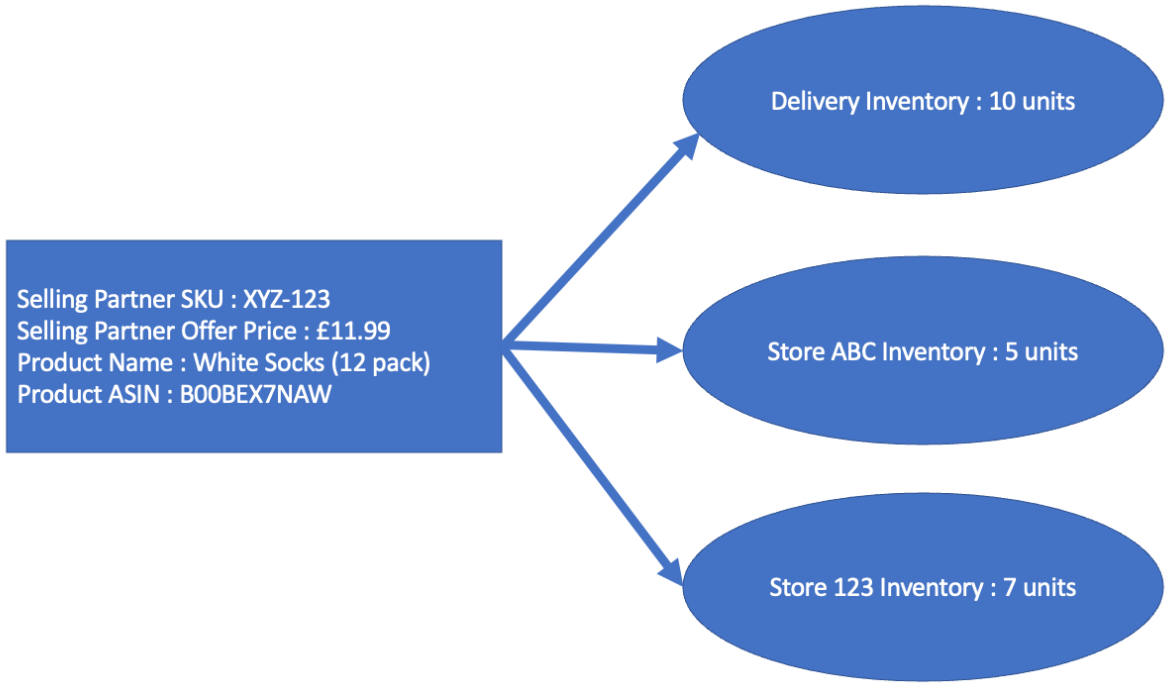
## Introduction

The following document provides a guide to implementing and managing BOPIS Store Inventory through the Amazon Selling Partner API.

## Overview

BOPIS is an extension of the Amazon Merchant Fulfilled Network (MFN). MFN is Amazon's terminology for Fulfilled by Merchant (FBM). It refers to a method of selling where you list products in Amazon stores, but you manage all storage, shipping, and customer support independently.

With BOPIS, you re-use the same SKU, Product Details and Pricing configured for FBM, sending your own store-level inventory to Amazon for each SKU. Your SKUs will now have multiple inventory values associated with them, representing your delivery inventory and inventory for each of your stores. Amazon maintains each stock level independently - if you sell an item for delivery, this does not change the stock level for any of your stores.




## Managing BOPIS Store Inventory

There are 2 ways to manage BOPIS Store Inventory through the Selling Partner API.

1) Using the Listings Items API to create, edit, delete and retrieve your BOPIS Store Inventory, on a per SKU basis.
2) Submitting a `JSON_LISTINGS_FEED` using the Feeds API to update inventory in bulk for multiple SKUs.

**WHAT IS THE LISTINGS ITEMS API?**

Using the Selling Partner API for Listings Items (Listings Items API), you can create, edit, delete, and retrieve details about your Amazon listings (SKUs). This includes product facts, such as item titles, and sales terms, such as price and inventory. See the Listings Items API Reference for details about Listings Items API operations and associated data types and schemas.

For the purposes of BOPIS, you can use the Listings Items API to manage the store-level inventory for your Amazon listings.

The Listings Items API accepts listings updates one at a time. For use-cases better suited to bulk uploads, the `JSON_LISTINGS_FEED` feed type may be used with the Selling Partner API for Feeds. The `JSON_LISTINGS_FEED` is the bulk equivalent of the Listings Items API, offering the same features and schemas provided by the Selling Partner API for Product Type Definitions.

**UPDATING STORE-LEVEL INVENTORY USING THE LISTINGS ITEMS API**

Using the **patchListingsItem** operation, you can update store-level inventory, one SKU per request. Within the same API call, you can also update delivery inventory for the SKU, should you choose to. Alternatively, if you have a separate process for updating delivery inventory (e.g. a separate feed or application, or manually via Seller Central), you can continue to run this and use **patchListingsItem** for store-level inventory only.

**Note:** The parameters associated with this operation may contain special characters that must be encoded to successfully call the API. To avoid errors with SKUs when encoding URLs, refer to URL Encoding.

**Usage Plan:**

You can make up to 5 requests per second for the **patchListingsItem** operation, with a burst capacity of 10.

| Rate (requests per second) | Burst |
|---|---|
| 5 | 10 |

The `x-amzn-RateLimit-Limit` response header returns the usage plan rate limits that were applied to the requested operation, when available. The table above indicates the default rate and burst values for this operation. Selling partners whose business demands require higher throughput may see higher rate and burst values than those shown here. For more information, see Usage Plans and Rate Limits in the Selling Partner API.

### Parameters

| Type | Name | Description | Schema |
|---|---|---|---|
| **Path** | **sellerId**<br>*required* | A selling partner identifier, such as a merchant account or vendor code. | string |
| **Path** | **sku**<br>*required* | A selling partner provided identifier for an Amazon listing. | string |
| **Query** | **marketplaceIds**<br>*required* | A comma-delimited list of Amazon marketplace identifiers for the request. | < string > array(csv) |
| **Query** | **issueLocale**<br>*optional* | A locale for localization of issues. When not provided, the default language code of the first marketplace is used. Examples: "en_US", "fr_CA", "fr_FR". Localized messages default to "en_US" when a localization is not available in the specified locale. | string |

| | | | |
|---|---|---|---|
| **Body** | **body**<br>*required* | The request body schema for the patchListingsItem operation. | ListingsItemPatchRequest |

### Header Parameters

| Name | Value |
|---|---|
| **Accepts** | application/json |
| **Content Type** | application/json |
| **x-amz-access-token** | *{your access token}* |

### Example - PATCH operation

The following example shows a PATCH operation for a single SKU to update inventory for 2 stores and the delivery inventory. Where a store's inventory is being updated, provide the Supply Source ID for that store as the **fulfillment_channel_code.** If you also intend to update the delivery inventory, then set the **fulfillment_channel_code** to **DEFAULT.**

With a PATCH operation, you only need to provide the fulfillment availability values where a change is to be applied. If you omit a store or the delivery inventory from your request body, any existing value held by Amazon will not be altered. This means you can manage the delivery inventory independently of the store inventory (e.g. via a separate feed), for the same SKU.

*Example PATCH operation to update inventory for 2 stores and the delivery ("DEFAULT") inventory*

```
PATCH /listings/2021-08-01/items/{sellerId}/{sku}?marketplaceIds={marketplaceIds}
&issueLocale={issueLocale}

{
    "productType": "PRODUCT",
    "patches": [{
        "op": "replace",
        "path": "/attributes/fulfillment_availability",
        "value": [{
                "fulfillment_channel_code": "c97b8f7a-9d5a-48ae-9f8a-c413a1b2c3d4",
                "quantity": 5

            },
            {
                "fulfillment_channel_code": "e1d86e7b-fe95-46e5-8b06-a7c8a5b6c7d8",
                "quantity": 7

            },
            {
                "fulfillment_channel_code": "DEFAULT",
                "quantity": 10
```

```
                }
            ]
        }]
}
```

### RETRIEVING STORE-LEVEL INVENTORY USING THE LISTINGS ITEMS API

Using the **getListingsItem** operation, you can retrieve inventory data, one SKU per request. It is not possible to retrieve store inventory data in bulk from Selling Partner API. The **getListingsItem** operation can return the full details for your listing including attributes, offers and issues. You can control the level of detail returned by the operation using the **includedData** parameter. To retrieve just the inventory data, set **includedData** to be **fulfillmentAvailability**.

**Note:** The parameters associated with this operation may contain special characters that must be encoded to successfully call the API. To avoid errors with SKUs when encoding URLs, refer to **URL Encoding**.

#### Usage Plan:

You can make up to 5 requests per second for the **getListingsItem** operation, with a burst capacity of 10.

| Rate (requests per second) | Burst |
|---|---|
| 5 | 10 |

#### Parameters

| Type | Name | Description | Schema |
|---|---|---|---|
| **Path** | **sellerId**<br>*required* | A selling partner identifier, such as a merchant account or vendor code. | string |
| **Path** | **sku**<br>*required* | A selling partner provided identifier for an Amazon listing. | string |
| **Query** | **marketplaceIds**<br>*required* | A comma-delimited list of Amazon marketplace identifiers for the request. | < string > array(csv) |
| **Query** | **issueLocale**<br>*optional* | A locale for localization of issues. When not provided, the default language code of the first marketplace is used. Examples: "en_US", "fr_CA", "fr_FR". Localized messages default to "en_US" when a localization is not available in the specified locale. | string |
| **Query** | **includedData**<br>*optional* | A comma-delimited list of data sets to include in the response. Default: summaries. | < enum (IncludedData) > array(csv) |

#### Header Parameters

| Name | Value |
|---|---|
| **Accepts** | application/json |
| **Content Type** | application/json |
| **x-amz-access-token** | *{your access token}* |

*Example request for **getListingsItem** operation, with includedData set to fulfillmentAvailability*

```
GET /listings/2021-08-01/items/{sellerId}/{sku}?marketplaceIds={marketplaceIds}
```

```
&issueLocale={issueLocale}&includedData=fulfillmentAvailability
```

*Example response for **getListingsItem** operation*

```
{
    "sku": "MY-PRODUCT-SKU",
    "fulfillmentAvailability": [{
            "fulfillmentChannelCode": "DEFAULT",
            "quantity": 10
        },
        {
            "fulfillmentChannelCode": "c97b8f7a-9d5a-48ae-9f8a-c413a1b2c3d4",
            "quantity": 5
        },
        {
            "fulfillmentChannelCode": "e1d86e7b-fe95-46e5-8b06-a7c8a5b6c7d8",
            "quantity": 7
        }
    ]
}
```

**USING THE SELLING PARTNER FEEDS API TO UPDATE STORE-LEVEL INVENTORY**

With **Selling Partner API for Feeds** (Feeds API), you can build applications that enable sellers to upload information to Amazon that helps them manage their selling businesses. There are feeds for a wide variety of use cases, such as creating listings, managing inventory and prices, acknowledging orders, and more. See Feed Type Values for a list of available feed types.

Via the **JSON_LISTINGS_FEED** feed type, the **Selling Partner API for Feeds** provides a way for you to make bulk updates (i.e. multiple SKUs / multiple stores) to your store-level inventory. You can supply multiple SKU level messages within the feed. Each message follows the same structure as for the Listings Items API **patchListingsItem** operation, but with the addition of a **messageId,** a **sku** and an **operationType** attribute. The **JSON_LISTINGS_FEED** schema can be found **here**, while the message schema can be found **here.**

**Please note -** there is a limit of 10K messages or 250MB per feed whichever is reached first.

*Example of a feed containing 2 messages for 2 SKUs ("SKU-10001" and "SKU-10002")*

```
{
    "header": {
        "sellerId": "{{merchant_id}}",
        "version": "2.0",
        "issueLocale": "{{issue_locale}}"
    },
    "messages": [{
            "messageId": 1,
            "sku": "SKU-10001",
            "operationType": "PATCH",
            "productType": "PRODUCT",
```

```
            "patches": [{
                "op": "replace",
                "path": "/attributes/fulfillment_availability",
                "value": [{
                        "fulfillment_channel_code": "a1b2c3...",
                        "quantity": 10
                    },
                    {
                        "fulfillment_channel_code": "x2y2z2...",
                        "quantity": 7
                    }
                ]
            }]
        },
        {
            "messageId": 2,
            "sku": "SKU-10002",
            "operationType": "PATCH",
            "productType": "PRODUCT",
            "patches": [{
                "op": "replace",
                "path": "/attributes/fulfillment_availability",
                "value": [{
                        "fulfillment_channel_code": "a1b2c3...",
                        "quantity": 9
                    },
                    {
                        "fulfillment_channel_code": "DEFAULT",
                        "quantity": 0
                    }
                ]
            }]
        }
    ]
}
```

**USING THE FEEDS API TO SUBMIT A JSON_LISTINGS_FEED**

To submit a JSON_LISTINGS_FEED through the Selling Partner API for Feeds, see the workflow below :

1. Call the `createFeedDocument` operation, specifying the content type for the feed that you are submitting. Amazon returns a `feedDocumentId` value and a URL for uploading the feed contents.

   You can make up to 0.5 requests per second for the `createFeedDocument` operation, with a burst capacity of 15.

| Rate (requests per second) | Burst |
|---|---|
| 0.5 | 15 |

*Example Request body for createFeedDocument operation, when submitting a JSON_LISTING_FEED*

```
{
    "contentType": "application/json; charset=UTF-8"
```

```
}
```

2. Upload your feed document contents to the URL from the previous step. Make sure to set the **Content-Type** header to have the same value that you submitted in step 1.
3. Call the `createFeed` operation. Use the `inputFeedDocumentId` parameter to pass in the `feedDocumentId` value from step 1. Specify the marketplaces that you want the feed to be applied to and any relevant feed options. Amazon returns a `feedId` value.

   You can make up to 0.0083 requests per second for the `createFeed` operation, with a burst capacity of 15.

| Rate (requests per second) | Burst |
|---|---|
| 0.0083 | 15 |

*Request structure for **createFeed** operation for a JSON_LISTINGS_FEED*

```
{
    "feedType": "JSON_LISTINGS_FEED",
    "marketplaceIds": ["{{marketplace_id}}"],
    "inputFeedDocumentId": "{{feedDocumentId}}"
}
```

4. Either :
   a. Periodically poll the Amazon SQS queue for the `FEED_PROCESSING_FINISHED` notification event, which provides information when the feed processing is `CANCELLED`, `DONE`, or `FATAL`. See the Notifications section below for how to set up for receiving `FEED_PROCESSING_FINISHED` notification events.
   b. Or, periodically poll the feed processing state using the **getFeed** operation, passing in the `feedId` returned at step 3.

      You can make up to 2 requests per second for the **getFeed** operation, with a burst capacity of 15.

| Rate (requests per second) | Burst |
|---|---|
| 2 | 15 |

5. Amazon returns the `resultFeedDocumentId` value in the notification when the feed moves into the `DONE` state.
6. Call the `getFeedDocument` operation. Use the `feedDocumentId` parameter to pass in the `resultFeedDocumentId` value from the previous step.

   You can make up to 0.0222 requests per second for the `getFeedDocument` operation, with a burst capacity of 10.

| Rate (requests per second) | Burst |
|---|---|
| 0.0222 | 10 |

   Amazon returns the `feedDocumentId` value, a URL for downloading the feed processing report, and the compression algorithm.

7. Download the feed processing report.
8. Check the feed processing report for errors generated during feed processing. If there are errors, correct them and submit the corrected feed, starting at step 1. If there are no errors, your feed submission was successful.

You can find the schema for the feed processing report here and an example feed processing report here.

For more details about submitting a feed, see Tutorial: Submit a feed.

## Notifications

The **Selling Partner API for Notifications** lets you subscribe to notifications from Amazon that are relevant to your business. Using this API, you can create a destination to receive notifications, subscribe to notifications, delete notification subscriptions, and more. Instead of polling for information, your application can receive information directly from Amazon when an event triggers a notification to which you are subscribed.

Depending on the type, notifications can be received through **Amazon EventBridge** or **Amazon Simple Queue Service (Amazon SQS)**:

- **Amazon EventBridge**: A serverless event bus that connects application data from your own applications, integrated Software-as-a-Service (SaaS) applications, and AWS services. For more information, refer to Amazon EventBridge.
- **Amazon Simple Queue Service (Amazon SQS)**: A fully managed message queuing service for microservices, distributed systems, and serverless applications. For more information, refer to Amazon Simple Queue Service.

The following Notifications can be useful for your BOPIS integration. These are received through **Amazon SQS:**

ORDER_STATUS_CHANGE - sent whenever there is a change in the status of new or existing order availability.

FEED_PROCESSING_FINISHED - sent whenever any feed submitted using the Selling Partner API for Feeds reaches a feed processing status of DONE, CANCELLED, or FATAL.

REPORT_PROCESSING_FINISHED - sent whenever any report that you have requested using the the Selling Partner API for Reports reaches a report processing status of DONE, CANCELLED, or FATAL.

You can find the complete list of available notification types here (Amazon SQS) and here (Amazon EventBridge).

**Setting up to receive Notifications (Amazon SQS)**

**Pre-requisites** : Login to your AWS Management Console, navigate to the Amazon SQS service, then create a Standard queue. If you are not familiar with Amazon SQS, this AWS SQS Getting Started guide provides more detail.

**Step 1** : Grant the Selling Partner API permission to write to your Amazon SQS queue. Follow the steps here to do this.

**Step 2** : Call the **createDestination** operation to create an Amazon Simple Queue Service (SQS) destination. Follow the steps here to complete this. This tells the Selling Partner API about your SQS queue. Note that **createDestination** is a Grantless Operation.

*Example createDestination operation*

```
POST /notifications/v1/destinations
{
    "name": "MyDestinationName",
    "resourceSpecification": {
        "sqs": {
```

```
            "arn": "arn:aws:sqs:eu-west-1:123456780000:MySQSQueueName"
        }
    }
}
```

*Example createDestination response*

```
{
    "payload": {
        "resource": {
            "sqs": {
                "arn": "arn:aws:sqs:eu-west-1:123456780000:MySQSQueueName"
            },
            "eventBridge": null
        },
        "destinationId": "a9a36dd9-6458-4329-9f42-ed4ecxxyyzzz",
        "name": "MyDestinationName"
    }
}
```

**Step 3** : Create a subscription to a notification type, to be delivered to the destination that you created in the previous step, using the **createSubscription** operation.

*Example createSubscription operation for ORDER_STATUS_CHANGE notifications*

```
POST /notifications/v1/subscriptions/ORDER_STATUS_CHANGE
{
  "payloadVersion": "1.0",
  "destinationId": "a9a36dd9-6458-4329-9f42-ed4ecxxyyzzz",
}
```

*Example createSubscription response*

```
{
    "payload": {
        "subscriptionId": "0c279d70-f5df-4b5b-985e-xy8fdaabbccc",
        "destinationId": "a9a36dd9-6458-4329-9f42-ed4ecxxyyzzz",
        "payloadVersion": "1.0"
    }
}
```

**Notification Structure**

Selling Partner notifications are in JSON format. Each notification contains a **Payload** object, which contains the actionable data of the notification. Notification Type, in combination with **PayloadVersion**, determines the structure of the **Payload** object.

A Selling Partner notification with **NotificationVersion**=1.0 contain the following properties:

| Object | Description | Type |
| --- | --- | --- |
| **NotificationVersion** | The notification version. This controls the structure of the notification. | string |
| **NotificationType** | The notification type. **NotificationType**, combined with **PayloadVersion,** controls the structure of the **Payload object**. | string |
| **PayloadVersion** | The payload version. **PayloadVersion**, combined with **NotificationType,** controls the structure of the **Payload object**. | string |
| **EventTime** | The date and time (in UTC) that the event which triggered the notification occurred. | string |
| **Payload** | The actionable data of the notification. The structure of the **Payload** is determined by **NotificationType**, in combination with **PayloadVersion**. | JSON object<br>For more information, see Notifications. |
| **NotificationMetadata** | The notification metadata. This includes the following objects:<br><br>**ApplicationId** – The identifier for the application that uses the notifications. Type = string<br>**SubscriptionId** - A unique identifier for the subscription which resulted in this notification. Type = string<br>**PublishTime** - The date and time (in UTC) that the notification was sent. Type = string<br>**NotificationId** - A unique identifier for this notification instance. Type = string | JSON object |

*Example FEED_PROCESSING_FINISHED notification*

```
{
    "notificationVersion": "2020-09-04",
    "notificationType": "FEED_PROCESSING_FINISHED",
    "payloadVersion": "2020-09-04",
    "eventTime": "2020-07-13T19:42:04.284Z",
    "payload": {
        "feedProcessingFinishedNotification": {
            "sellerId": "A3TH9S8BXXYYZZ",
            "feedId": "53347012345",
            "feedType": "JSON_LISTINGS_FEED",
            "processingStatus": "DONE",
            "resultFeedDocumentId": "amzn1.tortuga.3.edbcd0d8-3434-..."
        }
    },
    "notificationMetadata": {
        "applicationId": "amzn1.sellerapps.app.aacccfff-44aa-4b7c-b42b-xy8fdaabbccc",
        "subscriptionId": "subscription-id-d0e9e693-c3ad-4373-979f-ed4ecxxyyzzz",
        "publishTime": "2020-07-13T19:42:04.284Z",
        "notificationId": "d0e9e693-c3ad-4373-979f-ed4ec98xy123"
    }
}
```