Feeds API Use Case Guide

Version: 1.0

Date: 6/4/2020

Contents

What is the Feeds API?

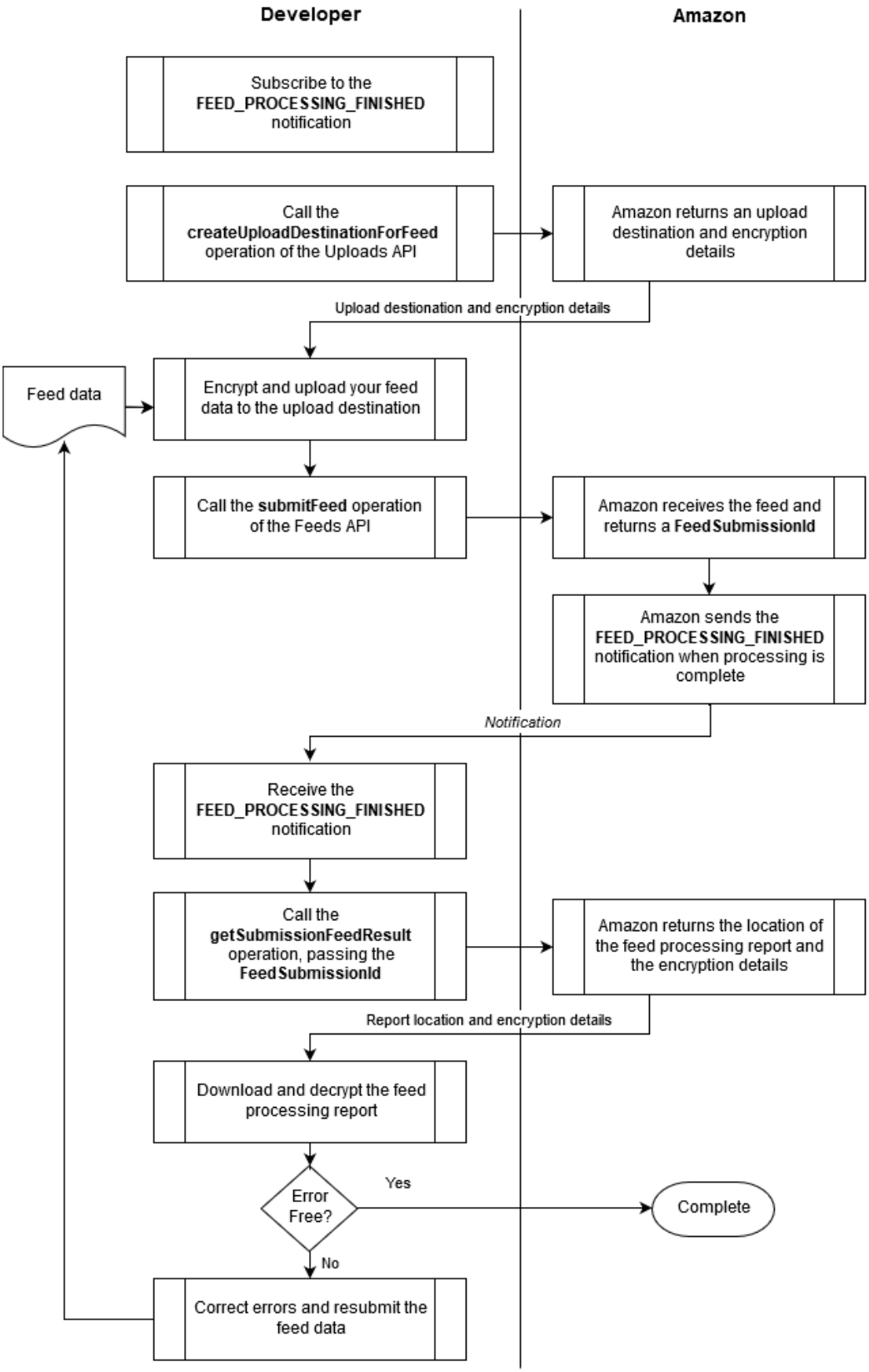
With the Selling Partner API for Feeds (Feeds API), you can build applications that enable sellers to upload information to Amazon that helps them manage their selling business. There are feeds for a wide variety of use cases, such as creating listings, managing inventory and prices, acknowledging orders,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

Workflow for submitting a feed

Submitting a feed is a multi-step process that includes calls to operation in both the Feeds API and the Uploads API. Here is the recommended workflow for submitting a feed:

1. Subscribe to the **FEED_PROCESSING_FINISHED** notification. This is a one-time task.
2. Call the **createUploadDestinationForFeed** operation of the Uploads API, specifying the feed type, content type, and content length for the feed that you are submitting.
3. Amazon returns an upload destination and feed encryption details.
4. Encrypt and upload your feed data to the upload destination.
5. Call the **submitFeed** operation of the Feeds API, specifying the type of feed you are submitting, the SHA256 hash of the feed content, the upload destination ID, and several optional parameters.
6. Amazon returns a **FeedSubmissionId** value.
7. Wait for the **FEED_PROCESSING_FINISHED** notification.
8. Amazon sends you the **FEED_PROCESSING_FINISHED** notification, indicating that feed processing is complete.
9. Call the **getFeedSubmissionResult** operation of the Feeds API, specifying the **FeedSubmissionId** value from step 6.
10. Amazon returns the location of the feed processing report along with the encryption details.
11. Download and decrypt the feed processing report.
12. Check the feed processing report for errors. If there are errors, correct them and resubmit the feed, starting at step 2.

Here is the recommended workflow for submitting a feed:



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

Terminology

- **Cipher block chaining**. Cipher block chaining is an algorithm that uses a block cipher to provide information security such as confidentiality or authenticity. This algorithm uses an initialization vector and a key to encrypt the data.
- **S3 presigned URL**. A URL for an AWS S3 bucket to which you can upload an object without AWS security credentials or permissions. You get an S3 presigned URL in Task 1. Create an upload destination.

Tutorial: Submit and verify a feed

This tutorial shows you how to submit a feed and verify that your feed submission was successful. Submitting a feed is comprised of three subtasks: creating an upload destination, encrypting and uploading your feed, and then calling the **submitFeed** operation. After submitting your feed, you must wait for Amazon to finish processing it before proceeding to the next step. You can use the **FEED_PROCESSING_FINISHED** notification to get confirmation that we have processed your feed. When feed processing is complete, you download and decrypt a feed processing report. Use the report to check for feed processing errors and resubmit your feed with corrections if necessary.

Prerequisites

To complete this tutorial, you will need:

1. A feed to submit. See FeedType enumeration for a list of the available feed types.
2. Authorization from the seller for whom you are making calls. See the Selling Partner API Developer Guide for more information.
3. A working Java Development Kit (JDK) installation, including the javax.crypto library.
4. An understanding of client-side encryption using the cipher block chaining (CBC). For definitions, see Terminology.

Java code samples

This tutorial contains Java code samples that can help you build a Java application that submits feeds to Amazon. You can use principles demonstrated in these code samples to guide you in building applications in other programming languages.

Steps

Step 1. Submit a feed

Step 2. Confirm feed processing

Step 3. Get the feed processing report

Step 4. Check the feed processing report for errors

Supplemental Java code

This section contains the `UploadDetails` class, the `EncryptionDetails` class, and the `CryptoProvider` interface, which are referenced in the sample Java code in Task 2. Encrypt and upload the feed data and Task 2. Download and decrypt the feed processing report.

```
/*UploadDetails.java*/
```

```
package com.amazon.spapi;

public final class UploadDetails
{
  private final String uploadDestinationId;
  private final String sha256Sum;
  public UploadDetails(String uploadDestinationId, String sha256Sum)
  {
    this.uploadDestinationId = uploadDestinationId;
    this.sha256Sum = sha256Sum;
  }

  public String getSha256Sum()
  {
    return sha256Sum;
  }

  public String getUploadDestinationId()
  {
    return uploadDestinationId;
  }
}
/*EncryptionDetails.java*/
package com.amazon.spapi;

public final class EncryptionDetails
{
  private final String key;
  private final String iv;
  public EncryptionDetails(String key, String iv)
  {
    this.key = key;
    this.iv = iv;
  }

  public String getKey()
  {
    return key;
  }

  public String getIv()
  {
    return iv;
  }
}

/*CryptoProvider.java*/
package com.amazon.spapi;

import javax.crypto.Cipher;
```

```
 @ FunctionalInterface
public interface CryptoProvider
{
  Cipher getInitializedCipher(int mode, EncryptionDetails
encryptionDetails);
}
```

Step 1. Submit a feed

Submitting a feed includes creating a destination, encrypting and uploading feed data, and calling the **submitFeed** operation of the Feeds API.

Tasks

Task 1. Create an upload destination

Task 2. Encrypt and upload the feed data

Task 3. Call the submitFeed operation

Task 1. Create an upload destination

Call the **createUploadDestinationForFeed** operation of the Uploads API to create an upload destination for the feed, in the form of an Amazon S3 presigned URL.

1. Call the **createUploadDestinationForFeed** operation of Uploads API, passing the following path and query parameters:

Path parameters:

Name	Description	Required
feedType	The type of feed that you want to upload to the destination. For **feedType** values, see FeedType enumeration. Type: string	Yes

Query parameters:

Name	Description	Required
Content-Type	The content type of the feed. For more information, see Setting the Content-Type value for a feed. Type: string	Yes
Content-Length	The content length of the feed in bytes. Type: int	Yes

Request example:

```
POST https://sellingpartnerapi-
na.amazon.com/uploads/v1/uploadDestinations/feeds/_POST_PRODUCT_DATA_?
Content-Type=text/tab-separated-values&Content-Length=256
```

Response

A successful response includes the following elements:

Name	Description	Required
uploadDestinationId	The unique identifier for referencing the upload destination. Type: string	Yes
url	The URL to upload the file to. Type: string	Yes
encryptionDetails	Encryption details for encrypting the feed. Type: EncryptionDetails	Yes

Response example:

```
{
  "uploadDestinationId": "3d4e42b5-1d6e-44e8-a89c-2abfca0625bb",
  "url": "https://s3.amazonaws.com/buyer-seller-messaging-test-draft-
attachment-namarketplace/%2F067/5d4e42b5-1d6e-44e8-a89c-
2abfca0625bb?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20190701T214102Z&X-Amz-SignedHeaders=content-md5%3Bhost%3Bx-amz-
server-side-encryption&X-Amz-Expires=900&X-Amz-
Credential=AKIAW5VUA47ENEOYT7RC%2F20190701%2Fus-east-
1%2Fs3%2Faws4_request&X-Amz-
Signature=d4f85c5f1a32a788a8d54e3f00a2a08af45be5b83551cdd81c82ae353dfc
dfd4",
  "encryptionDetails":
  {
    "standard": "AES",
    "initializationVector": "8f6ccc560d50a2d031ec80bef26a1d0a",
    "key": "key"
  }
}
```

2. Save the following values:
    - **initializationVector**, **key**, and **url**. Use these values in Task 2. Encrypt and upload the feed data.
    - **uploadDestinationId**. Use this value in Task 3. Call the submitFeed operation.

Task 2. Encrypt and upload the feed data

The Java sample code in this task contains logic for encrypting and uploading a feed. This sample code uses the Apache HTTP client. See Supplemental Java code for the types referenced in the sample code.

1. Use the following as input for the sample code:
    - Your feed data is the argument for the `stream` parameter of the `InputStream` method of the `UploadToDestinationExample` class.
    - The **initializationVector** and **key** values that you saved in Task 1. Create an upload destination are arguments for the `iv` and `key` parameters of the `EncryptionDetails` method of `EncryptionDetails` class.
    - The **url** value that you saved in Task 1. Create an upload destination is the argument for the `url` parameter of the `uploadToDestination` method of the `UploadToDestinationExample` class.
2. Save the `sha256sum` value to pass in with the **ContentSha256** parameter in Task 3. Call the submitFeed operation.

Sample Java code

```
package com.amazon.spapi;

import org.apache.commons.io.IOUtils;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.entity.InputStreamEntity;
import org.apache.http.impl.client.HttpClients;

import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.IOException;
import java.io.InputStream;
import java.security.DigestInputStream;
import java.security.GeneralSecurityException;
import java.security.Key;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.Base64;

public class UploadToDestinationExample
{

  static final String SHA_256 = "SHA-256";
  static final String AES = "AES";
  static final int AES_BLOCK_SIZE = 16;
  static final Base64.Encoder BASE64_ENCODER = Base64.getEncoder();
  static final Base64.Decoder BASE64_DECODER = Base64.getDecoder();
```

```
  static final CryptoProvider AES_CRYPTO_PROVIDER =
UploadToDestinationExample::getInitializedCipher;

  static InputStream buildCipherInputStream(EncryptionDetails
encryptionDetails, InputStream stream, int mode)
  {
    return new CipherInputStream(stream,
      AES_CRYPTO_PROVIDER.getInitializedCipher(mode,
encryptionDetails));
  }

  private static InputStream buildCipherInputStream(EncryptionDetails
encryptionDetails, InputStream stream)
  {
    return buildCipherInputStream(encryptionDetails, stream,
Cipher.ENCRYPT_MODE);
  }

  static Cipher getInitializedCipher(int mode, EncryptionDetails
details)
  {

    Cipher cipher;
    try
    {
      cipher = Cipher.getInstance(AES);
      Key key = new
SecretKeySpec(BASE64_DECODER.decode(details.getKey()), AES);
      byte[]iv = BASE64_DECODER.decode(details.getIv());
      IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);
      cipher.init(mode, key, ivParameterSpec, new SecureRandom());
    }
    catch (GeneralSecurityException e)
    {
      throw new IllegalStateException("Could not create Cipher for
key-iv pair", e);
    }

    return cipher;
  }

  UploadDetails uploadToDestination(EncryptionDetails
encryptionDetails, String uploadDestinationId, String url,
    String contentType,
    long documentLength,
    InputStream inputStream)
  {
    try
    {
```

```
      inputStream = buildCipherInputStream(encryptionDetails,
inputStream);
      DigestInputStream sha256sumStream = new
DigestInputStream(inputStream, MessageDigest.getInstance(SHA_256));
      inputStream = sha256sumStream;

      // Put the file
      HttpClient httpClient = HttpClients.createDefault();
      HttpPut httpPut = new HttpPut(url);
      // This content length calculation is specific to AES
      long contentLength = (documentLength / AES_BLOCK_SIZE + 1) *
AES_BLOCK_SIZE;
      // This sets the Content-Length header since we specified the
contentLength
      HttpEntity document = new InputStreamEntity(inputStream,
contentLength);
      httpPut.setHeader("Content-Type", contentType);
      httpPut.setEntity(document);
      //Put the content in the S3 Pre-signed URL
      HttpResponse httpResponse = httpClient.execute(httpPut);
      if (httpResponse == null ||
(httpResponse.getStatusLine().getStatusCode() / 100) != 2)
      {
        // Handle error responses here.
        throw new IllegalStateException("Could not upload to S3.");
      }
      // Document was successfully uploaded!
      byte[]sha256sumDigest =
sha256sumStream.getMessageDigest().digest();
      String sha256sum =
BASE64_ENCODER.encodeToString(sha256sumDigest);
      return new UploadDetails(uploadDestinationId, sha256sum);
    }
    catch (IOException | NoSuchAlgorithmException e)
    {
      throw new RuntimeException("Error occurred when attempting to
encrypt or upload to S3.", e);
    }
    finally
    {
      IOUtils.closeQuietly(inputStream);
    }
  }
}
```

Task 3. Call the submitFeed operation

Call the **submitFeed** operation of the Feeds API to specify the type of feed you are submitting, the SHA256 hash of the feed content, the upload destination ID, and any optional parameters that you want.

1. Call the **submitFeed** operation, passing the following parameters:

Path parameters:

Name	Description	Required
feedType	The type of feed that you are submitting. For **feedType** values, see FeedType enumeration. Type: string	Yes

Body parameters:

Name	Description	Required
feedSubmission	Request schema for submitting a feed. Type: SubmitFeedRequest	Yes

Request example:

```
POST https://sellingpartnerapi-
na.amazon.com/feeds/v0/submit/{feedType}
{
  "MarketplaceIds": [
    "ATVPDKIKX0DER",
    "A1F83G8C2ARO7P"
  ],
  "ContentSha256":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "UploadDestinationId": "73487458345"
}
```

Request example for an Easy Ship order:

```
POST https://sellingpartnerapi-
na.amazon.com/feeds/v0/submit/_POST_EASYSHIP_DOCUMENTS_
{
  "MarketplaceIds": ["A21TJRUUN4KGV"],
  "ContentSha256":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "UploadDestinationId": "3d4e42b5-1d6e-44e8-a89c-2abfca0625bb",
  "FeedOptions":
  {
    "AmazonOrderId": "902-3159896-1390916",
```

```
    "DocumentType": "ShippingLabel"
  }
}
```

Response
A successful response includes the following elements:

Name	Description	Required
FeedSubmissionId	A unique feed submission identifier. Type: string	Yes

Response example:
{
"FeedSubmissionId": "23492394"
}

2. Save the **FeedSubmissionId** value. Pass this value in with a call to the **getFeedSubmissionResult** operation in Task 1. Get location and encryption information.

Step 2. Confirm feed processing

After you call the submitFeed operation you need to wait for confirmation that we have processed your feed before you can continue. Amazon recommends subscribing to the **FEED_PROCESSING_FINISHED** notification to get this confirmation. After you subscribe, we will send you a push notification when we finish processing any feed that you submit. To subscribe to the **FEED_PROCESSING_FINISHED**, see the Notifications use case guide. Subscribing to the **FEED_PROCESSING_FINISHED** notification is a one-time task.

Note. An alternative way to confirm feed processing is to poll the **getFeedSubmissionList** operation until the response indicates that feed processing is complete. A downside to polling is that repeated calls the **getFeedSubmissionList** operation could make you exceed throttling limits. If polling is the better option for you, however, go to Step 2 (alternate). Confirm feed processing by polling.

To confirm feed processing with notifications

1. Be sure you are subscribed to the **FEED_PROCESSING_FINISHED** notification. To subscribe to this notification, see the Notifications use case guide.
2. After Task 3. Call the submitFeed operation, wait for the **FEED_PROCESSING_FINISHED** notification.

   When feed processing is complete, you receive the **FEED_PROCESSING_FINISHED** notification with the **FeedProcessingStatus** element set to one of these values:

   - *DONE* - The feed was successfully submitted. Use the **FeedSubmissionId** value included in the notification as input for the **getFeedSubmissionResult** operation in Task 1. Get location and encryption information.
   - *CANCELLED* - The feed was cancelled by the seller or by Amazon.

Step 2 (alternate). Confirm feed processing by polling

Important. Amazon recommends confirming feed processing using notifications. To do this, go to Step 2. Confirm feed processing.

An alternative way to confirm feed processing is to poll the **getFeedSubmissionList** operation of the Feeds API until the response indicates that feed processing is complete. A downside to polling is that repeated calls the **getFeedSubmissionList** operation could make you exceed throttling limits. If polling is the better option for you, however, use the following workflow.

Tasks

Task 1. Call the getFeedSubmissionList operation

Task 2. Check the feed processing status

Task 1. Call the getFeedSubmissionList operation

You can call the **getFeedSubmissionList** operation of the Feeds API to check the status of the feed that you submitted in Task 3. Call the submitFeed operation.

1. Call the **getFeedSubmissionList** operation, passing in the **FeedSubmisionId** value returned by the **submitFeed** operation in Task 3. Call the submitFeed operation.

Query parameters:

Name	Description	Required
FeedSubmissionIds	A list of no more than 100 **FeedSubmissionId** values. Type: array[string]	No
FeedTypes	A list of one or more **FeedType** values by which to filter the list of feed submissions. For **FeedType** values, see FeedType enumeration. Type: array[string]	No
MaxCount	A non-negative integer that indicates the maximum number of feed submissions to return in the list. Type: int	No
FeedProcessingStatuses	A list of one or more feed processing statuses by which to filter the list of feed submissions. For possible values, see FeedProcessingStatus enumeration. Type: array[string]	No
SubmittedFromDate	The earliest submission date to include, in ISO8601 date time format. Type: dateTime	No

Name	Description	Required
SubmittedToDate	The latest submission date to include, in ISO8601 date time format. Type: dateTime	No
NextToken	A string token returned in the response to your previous request. Type: string	

Request example:

```
GET https://sellingpartnerapi-
na.amazon.com/feeds/v0/submissions?FeedSubmissionIds=
FeedSubmissionIdExample1,FeedSubmissionIdExample2&MaxCount=10
```

Response

A successful response includes the following elements:

Name	Description	Required
NextToken	When **HasNext** is *true*, pass this string token in the next request to return the next response page. Type: string	No
HasNext	When *true*, pass the **NextToken** value in the next request to return the next response page. Type: boolean	No
FeedSubmissionInfoList	A list containing information about each feed submission. Type: list of FeedSubmissionInfoList	Yes

Response Example:

```
{
  "NextToken": "eukhdkdghdkjsfhjdfhkfjsd",
  "HasNext": true,
  "FeedSubmissionInfoList": [
    {
      "FeedSubmissionId": "FeedSubmissionId1",
      "FeedType": "_POST_PRODUCT_DATA_",
      "SubmitedDate": "2019-12-11T13:16:24.630Z",
      "FeedProcessingStatus": "_DONE_",
      "StartedProcessingDate": "2019-12-11T13:16:24.630Z",
      "CompletedProcessingDate": "2019-12-11T13:16:24.630Z"
```

```
      }
   ]
}
```

2. Go to Task 2. Check the feed processing status.

Task 2. Check the feed processing status

Check the value of **FeedProcessingStatus** property that was returned in Task 1. Call the getFeedSubmissionList operation.

- If **FeedProcessingStatus**=*_DONE_*, feed processing is complete. Go to Step 3. Get the feed processing report.
- If **FeedProcessingStatus**=*_IN_PROGRESS_*, feed processing is not yet complete. Go back to Task 1. Call the getFeedSubmissionList operation. Repeat until feed processing is complete.

Step 3. Get the feed processing report

The feed processing report indicates which records in the feed that you submitted were successful and which records generated errors. Get this report by first getting location and encryption information and then downloading and decrypting the report.

Tasks

Task 1. Get location and encryption information

Task 2. Download and decrypt the feed processing report

Task 1. Get location and encryption information

Call the **getFeedSubmissionResult** operation of Feeds API to get the location of your feed processing report and the information you will need to decrypt it.

1. Call the **getFeedSubmissionResult** operation using the following parameters:

Path parameters:

Name	Description	Required
FeedSubmissionId	The identifier for the feed submission that you saved in Task 3. Call the submitFeed operation. Type: string	Yes

Request example:

```
GET https://sellingpartnerapi-
na.amazon.com/feeds/v0/submissions/23492394/result
```

Response

A successful response includes the following elements:

Name	Description	Required
destination	The location of the feed processing report. Type: destination	Yes
encryptionDetails	Encryption details for decrypting the feed processing report data. Type: encryptionDetails	Yes
isGizipped	When *true*, the feed processing report is compressed using Gzip compression. Type: boolean	No

Response example:

```
{
  "destination":
  {
    "channel": "s3",
    "url": "https://s3.amazonaws.com/buyer-seller-messaging-test-
draft-attachment-namarketplace/%2F067/5d4e42b5-1d6e-44e8-a89c-
2abfca0625bb?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20190701T214102Z&X-Amz-SignedHeaders=content-md5%3Bhost%3Bx-amz-
server-side-encryption&X-Amz-Expires=900&X-Amz-
Credential=AKIAW5VUA47ENEOYT7RC%2F20190701%2Fus-east-
1%2Fs3%2Faws4_request&X-Amz-
Signature=d4f85c5f1a32a788a8d54e3f00a2a08af45be5b83551cdd81c82ae353dfc
dfd4"
  },
  "encryptionDetails":
  {
    "standard": "AES",
    "initializationVector": "58fabfa70811950fc1a8c6e0d56faec8",
    "key": "Sample"
  }
}
```

2. Save the **destination** and **encryptionDetails** values to pass in Task 2. Download and decrypt the feed processing report.

Task 2. Download and decrypt the feed processing report

The Java sample code in this task contains logic for downloading and decrypting the feed processing report. This sample code uses the Apache HTTP client. See Supplemental Java code for the types referenced in the sample code.

1. Use the following as inputs for the sample code:
   - The **destination** value that you saved in Task 1. Get location and encryption information is the argument for the `destination` parameter of the `downloadAndDecryptReportContent` method of the `DecryptFeedProcessingReportExample` class.
   - The **encryptionDetails** value that you saved in Task 1. Get location and encryption information is the argument for the `details` parameter of the `decryptFeedSubmissionReportContent` method of the `DecryptFeedProcessingReportExample` class.
2. Save the `resultStream` value. This is your decrypted feed processing report that you will check in Step 4. Check the feed processing report for errors.

Sample Java code

```
package com.amazon.spapi;

import io.swagger.client.model.EncryptionDetails;
import io.swagger.client.model.UploadDestination;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.HttpClients;

import javax.crypto.Cipher;
import java.io.IOException;
import java.io.InputStream;
import java.util.zip.GZIPInputStream;

import static
com.amazon.spapi.UploadToDestinationExample.buildCipherInputStream;

public class DecryptFeedProcessingReportExample
{

  public InputStream downloadAndDecryptReportContent(UploadDestination
destination, boolean isGzipped)
  throws IOException
  {
    InputStream result = null;
    HttpResponse httpResponse = null;
    String url = destination.getUrl();
    // Acquire the file
    HttpClient httpClient = HttpClients.createDefault();
    HttpGet httpGet = new HttpGet(url);
    httpResponse = httpClient.execute(httpGet);
    if (httpResponse == null ||
httpResponse.getStatusLine().getStatusCode() ==
HttpStatus.SC_NOT_FOUND)
```

```
    {
      throw new IllegalArgumentException("Could not find result at
destination.");
    }
    HttpEntity entity = httpResponse.getEntity();
    if (entity == null)
    {
      throw new RuntimeException("The HTTP store returned success but
no document.");
    }
    result = decryptFeedSubmissionReportContent(entity.getContent(),
destination.getEncryptionDetails(), isGzipped);
    return result;
  }

  private InputStream decryptFeedSubmissionReportContent(InputStream
input, EncryptionDetails details,
    boolean isGzipped)
  throws IOException
  {
    InputStream resultStream = input;
    // If encrypted, decipher the stream
    if (details != null &&
EncryptionDetails.StandardEnum.AES.equals(details.getStandard()))
    {
      com.amazon.spapi.EncryptionDetails encryptionDetails;
      encryptionDetails = new
com.amazon.spapi.EncryptionDetails(details.getKey(),
          details.getInitializationVector());
      resultStream = buildCipherInputStream(encryptionDetails,
resultStream, Cipher.DECRYPT_MODE);
    }

    // Determine if the stream should be unzipped as well
    if (isGzipped)
    {
      resultStream = new GZIPInputStream(resultStream);
    }

    return resultStream;
  }

}
```

Step 4. Check the feed processing report for errors

Check the feed processing report for errors. If there are no errors, your feed submission is complete. If there are errors, correct them and resubmit the feed, starting at Task 1. Create an upload destination. Repeat the process until there are no errors in the feed processing report.

Feed behavior

Feed processing

Here is some general feed processing behavior that you can expect:

- Inventory feeds (product, price, inventory, relationship, image, or override feeds) and order feeds are processed separately; they can be submitted simultaneously.
- _POST_PRODUCT_DATA_ feeds can be processed along with price, inventory, and other XML feeds. However, the price, inventory, and other feeds will fail if they refer to SKUs that the product feed hasn't finished processing. You should serialize price, inventory, and image updates after product feeds have completed.
- All inventory feeds, other than _POST_PRODUCT_DATA_, can be submitted at the same time. For example price, inventory availability, relationship, and image feeds can all be submitted at the same time.
- Feeds of the same type are processed sequentially. This applies to all inventory feed types. For example, if you submit two pricing feeds, only one is processed at a time.
- Optimize your feed submissions. Uploading many small feeds every few seconds is very inefficient and can result in a backlog, blocking other feeds from processing and forcing you to cancel some of the previously submitted feeds.

Submitting feeds with multiple MarketplaceId values

If a seller is registered in multiple marketplaces, then they have multiple **MarketplaceId** values associated with their **SellerId**. You can submit a feed, on a seller's behalf, that is applied to one or several **MarketplaceId** values. If you are in the Europe or North America region, you can submit feeds to support multiple marketplaces on behalf of a seller that has registered using a single, unified seller account. You can use the Sellers API section to determine what **MarketplaceId** values are associated with a specified **SellerId**.

Behavior of feeds when submitting multiple MarketplaceId values

If you include multiple **MarketplaceId** values when submitting a feed request, feed processing has a more complex behavior. The following are some general rules when submitting a feed request with multiple **MarketplaceId** values:

- Flat file feeds can only be applied to a single marketplace. For example, a flat file feed submission would not be accepted if the **MarketplaceId** values were for a DE seller account and an FR seller account.
- If more than one **MarketplaceId** value is submitted and one or more of those **MarketplaceId** values fail validation for whatever reason (currency mismatch, language mismatch, country mismatch, one blocked and the other valid), then Amazon returns an error and the submission fails.
- Amazon validates a feed submission before it can be queued for processing. A feed submission passes validation when it contains appropriate **MarketplaceId** values for the FeedType submitted. Passing validation does not mean that the feed is correctly formatted or that it will process successfully.
- When you call the **submitFeed** operation with **PurgeAndReplace** set to *true*, the purge will be applied to all EU or NA marketplaces specified.

Behavior of XML Product Feeds and XML Relationship Feeds when used with multiple MarketplaceId values

An XML Product Feed or XML Relationship Feed submission that specifies a list of **MarketplaceId** values must specify marketplaces that all share the same language code or the feed is rejected at submission time. If no **MarketplaceId** values are specified, the feed is applied to all marketplaces that the seller is registered in and that share the same language code as the seller's default marketplace. This behavior applies to the following feeds:

- **Product Feed** (_POST_PRODUCT_DATA_)
- **Relationships Feed** (_POST_PRODUCT_RELATIONSHIP_DATA_)

Behavior of XML Inventory Feeds when used with multiple MarketplaceId values

In EU (for all sellers) and in NA (for self-fulfilled sellers only), quantity is a global value in relationship with a SKU, so changes to stock levels are reflected in all marketplaces in which the SKU is active. If multiple XML Inventory Feeds are processed for the same SKU in different marketplaces, then the quantity of the last uploaded XML Inventory Feed from the seller reflects the global inventory level. Setting the item inventory level to 0 effectively sets the item quantity to 0 in all marketplaces and makes the item non-buyable. All listing information is still maintained in the system. This behavior applies to the following feed:

- **Inventory Feed** (_POST_INVENTORY_AVAILABILITY_DATA_)

Behavior of XML Overrides Feeds when used with multiple MarketplaceId values

Only a single marketplace can be specified for XML Overrides Feeds. This behavior applies to the following feed:

- **Overrides Feed** (_POST_PRODUCT_OVERRIDES_DATA_)

Behavior of XML Pricing Feeds when used with multiple MarketplaceId values

An XML Pricing Feed submission that specifies a list of **MarketplaceId** values must specify **MarketplaceId** values that all share the same currency code or the feed is rejected at submission time. If no **MarketplaceId** values are specified, the feed is applied to all marketplaces that the seller is registered in that share the same currency code as the seller's default marketplace. This behavior applies to the following feed:

- **Pricing Feed** (_POST_PRODUCT_PRICING_DATA_)

Behavior of XML Product Image Feeds when used with multiple MarketplaceId values

XML Product Image Feeds map images to ASINs in the provided marketplaces. If no **MarketplaceId** values are specified, the feed is applied to all marketplaces that the seller is registered in and that are in the same country as the seller's original marketplace registration. This behavior applies to the following feed:

- **Product Images Feed** (_POST_PRODUCT_IMAGE_DATA_)

Behavior of Flat File Product and Inventory Feeds when used with multiple MarketplaceId values

Flat File Product and Inventory Feeds can only be applied to one country. However, in EU (for all sellers) and in NA (for self-fulfilled sellers only), quantity is a global value in relationship with a SKU, so changes to stock levels are reflected in all marketplaces that the SKU is active in. If multiple inventory feeds are processed for the same SKU in different marketplaces, then the quantity of the last uploaded inventory feed from the seller reflects the global inventory level. Setting the item inventory level to 0 effectively sets the item quantity to 0 in all marketplaces and makes the item non-buyable. All listing information is still maintained in the system. This behavior applies to the following feeds:

- **Flat File Inventory Loader Feed** (_POST_FLAT_FILE_INVLOADER_DATA_)
- **Flat File Listings Feed** (_POST_FLAT_FILE_LISTINGS_DATA_)
- **Flat File Book Loader Feed** (_POST_FLAT_FILE_BOOKLOADER_DATA_)

- **Flat File Music Loader Feed** (_POST_FLAT_FILE_CONVERGENCE_LISTINGS_DATA_)
- **Flat File Video Loader Feed** (_POST_FLAT_FILE_LISTINGS_DATA_)
- **Flat File Price and Quantity Update Feed** (_POST_FLAT_FILE_PRICEANDQUANTITYONLY_UPDATE_DATA_)
- **UIEE Inventory Feed** (_POST_UIEE_BOOKLOADER_DATA_)

Behavior of Order Feeds when used with multiple MarketplaceId values

All Order Feeds refer to an Amazon order ID, which is a globally unique identifier. Therefore, Order Feeds are not marketplace-specific. This behavior applies to the following feeds:

- **Order Acknowledgement Feed** (_POST_ORDER_ACKNOWLEDGEMENT_DATA_)
- **Order Adjustments Feed** (_POST_PAYMENT_ADJUSTMENT_DATA_)
- **Order Fulfillment Feed** (_POST_ORDER_FULFILLMENT_DATA_)
- **Flat File Order Acknowledgement Feed** (_POST_FLAT_FILE_ORDER_ACKNOWLEDGEMENT_DATA_)
- **Flat File Order Adjustments Feed** (_POST_FLAT_FILE_PAYMENT_ADJUSTMENT_DATA_)
- **Flat File Order Fulfillment Feed** (_POST_FLAT_FILE_FULFILLMENT_DATA_)

Behavior of XML FBA Fulfillment Order Feeds when used with multiple MarketplaceId values

All XML FBA Fulfillment Order Feeds can only be applied to one country. This behavior applies to the following feeds:

- **FBA Fulfillment Order Feed** (_POST_FULFILLMENT_ORDER_REQUEST_DATA_)
- **FBA Fulfillment Order Cancellation Feed** (_POST_FULFILLMENT_ORDER_CANCELLATION_REQUEST_DATA_)

Error messages when submitting multiple MarketplaceId values

There are several error messages that can be returned when submitting requests with multiple **MarketplaceId** values:

Error Message	Description
All specified marketplaces for this feed type must have the same default language code. [ABCD], [EFGH] have different default language codes.	Some feeds, such as the _POST_PRODUCT_DATA_ feed, can only be applied to marketplaces that share the same language. The specified **MarketplaceId** values do not share the same language.
All specified marketplaces for this feed type must have the same default currency code. [ABCD], [EFGH] have different default currency codes.	Some feeds, especially those that deal with pricing such as the _POST_PRODUCT_PRICING_DATA_ feed, can only be applied to marketplaces that share the same currency. The specified **MarketplaceId** values do not share the same currency.
All specified marketplaces for this feed type must be based in the same country. [ABCD], [EFGH] have different default country codes.	Flat-file feeds can only be applied to **MarketplaceId** values that are registered in the same country. The specified **MarketplaceId** values do not share the same default country.

Error Message	Description
The specified marketplaces are correctly associated with your account, but you are prevented from performing this action in the following marketplaces: [ABCD], [ABCD]. Please contact Seller Support in your default marketplace for more information about your account.	There is some issue with your account and the **MarketplaceId** you specified. You can get this error message for several reasons, including not completing a marketplace registration. Contact Seller Support in your home marketplace to clear up the issue.
Your feed could not be applied to any marketplaces.	Since you did not provide a **MarketplaceId**, Amazon attempted to determine an appropriate marketplace to use. It was unable to find a marketplace associated with your account that could be used to fulfill your request.

Best practices

Setting the Content-Type value for a feed

Your feeds must be in a valid encoding based on marketplace and feed content type, and that encoding must be specified with the **Content-Type** parameter of the **createUploadDestinationForFeed** operation. For more information, see Task 1. Create an upload destination.

The following table shows the **Content-Type** values for flat file feeds and XML feeds, by marketplace:

Marketplace	Content-Type for flat file feeds	Content-Type for XML feeds
North America and Europe	text/tab-separated-values; charset=iso-8859-1	text/xml
Japan	text/tab-separated-values; charset=Shift_JIS	text/xml

Maximizing feed performance

You can generally get the best overall feed processing performance by following these guidelines:

- Avoid submitting a lot of feeds with only a few records in each feed. When possible, combine the data into larger feeds that you submit less frequently.
- Include only the products you are updating, not your entire inventory.
- Upload one feed of the same type no more than once every 20 minutes. Allow more time between larger feeds.
- Keep file size below 10 MiB ($5*2^{21}$, or 10,485,760 bytes).

FeedProcessingStatus enumeration

There are different types of processing status that are associated with a feed while it is being processed. These are all the feed processing status values that are available through the Feeds API.

Value	Description
_AWAITING_ASYNCHRONOUS_REPLY_	The request is being processed, but is waiting for external information before it can complete.
CANCELLED	The request has been aborted due to a fatal error.
DONE	The request has been processed. You can call the **getFeedSubmissionResult** operation, discussed in Step 3. Get the feed processing report, to receive a processing report that describes which records in the feed were successful and which records generated errors.
_IN_PROGRESS_	The request is being processed.
_IN_SAFETY_NET_	The request is being processed, but the system has determined that there is a potential error with the feed (for example, the request will remove all inventory from a seller's account.) An Amazon seller support associate will contact the seller to confirm whether the feed should be processed.
SUBMITTED	The request has been received, but has not yet started processing.
UNCONFIRMED	The request is pending.

FeedType enumeration

The **FeedType** enumeration indicates to Amazon how to process a feed that you submit. This section includes **FeedType** enumeration values for the various feed types.

Feed types fall into these categories:

- Product and inventory feeds
- Order feeds
- Fulfillment by Amazon feeds
- Business feed
- Easy Ship feed

Product and inventory feeds

Name	File
Product Feed Enumeration value: _POST_PRODUCT_DATA_	Product.xsd More information: Create Products - Product Feed Schema
Inventory Feed Enumeration value: _POST_INVENTORY_AVAILABILITY_DATA_	Inventory.xsd More information: Update Quantity Available - Inventory Feed Schema
Overrides Feed Enumeration value: _POST_PRODUCT_OVERRIDES_DATA_	Override.xsd More information: XSDs
Pricing Feed Enumeration value: _POST_PRODUCT_PRICING_DATA_	Price.xsd More information: Assign a Price - Price Feed Schema
Product Images Feed Enumeration value: _POST_PRODUCT_IMAGE_DATA_	ProductImage.xsd More information: Send Product Images - Image Feed Schema
Relationships Feed Enumeration value: _POST_PRODUCT_RELATIONSHIP_DATA_	Relationship.xsd More information: Establish Product Relationships - Relationship Feed Schema (not applicable to all categories)
Flat File Inventory Loader Feed Enumeration value: _POST_FLAT_FILE_INVLOADER_DATA_	Flat.File.InventoryLoader.xls- For creating or updating listings for products already in Amazon's catalog. More information: Using the Inventory Loader

Name	File
Flat File Listings Feed Enumeration value: _POST_FLAT_FILE_LISTINGS_DATA_	To create a listing for a product not yet in the Amazon catalog, use the corresponding category template file in Templates for Specific Categories You can also create and update listings for products already in Amazon's catalog using Flat.File.Listingloader.xls.
Flat File Book Loader Feed Enumeration value: _POST_FLAT_FILE_BOOKLOADER_DATA_	Flat.File.BookLoader.xls More information: Use Book Loader
Flat File Music Loader Feed Enumeration value: _POST_FLAT_FILE_CONVERGENCE_LISTINGS_DATA_	Flat.File.Music.xls More information: Use Music Loader
Flat File Video Loader Feed Enumeration value: _POST_FLAT_FILE_LISTINGS_DATA_	Flat.File.Video.xls More information: Use Video Loader
Flat File Price and Quantity Update Feed Enumeration value: _POST_FLAT_FILE_PRICEANDQUANTITYONLY_UPDATE_DATA_	Flat.File.PriceInventory.us.xls More information: Price & Quantity File
UIEE Inventory Feed Enumeration value: _POST_UIEE_BOOKLOADER_DATA_	standard-book-template.xls More information: UIEE for Books
ACES 3.0 Data (Automotive Part Finder) Feed Enumeration value: _POST_STD_ACES_DATA_	AutoAccessory.xsd More information: Provide Fitment Data to Amazon

Order feeds

Name	File
Order Acknowledgement Feed	OrderAcknowledgement.xsd

Enumeration value: _POST_ORDER_ACKNOWLEDGEMENT_DATA_	More information: Acknowledge Receipt of Orders - Order Acknowledgement
Order Adjustments Feed Enumeration value: _POST_PAYMENT_ADJUSTMENT_DATA_	OrderAdjustment.xsd More information: Refund or Partially Cancel Orders - Order Adjustment or Partial Cancellation
Order Fulfillment Feed Enumeration value: _POST_ORDER_FULFILLMENT_DATA_	OrderFulfillment.xsd More information: Confirm Shipment and Get Paid - Order Fulfillment
Invoice Confirmation Feed Enumeration value: _POST_INVOICE_CONFIRMATION_DATA_	InvoiceConfirmation.xsd More information: Invoice Confirmation
Sourcing On Demand Feed (Japan only) Enumeration value: _POST_EXPECTED_SHIP_DATE_SOD_	OrderSourcingOnDemand.xsd More information: Manage Orders with XML
Flat File Order Acknowledgement Feed This feed cancels orders. This feed does not acknowledge orders. Enumeration value: _POST_FLAT_FILE_ORDER_ACKNOWLEDGEMENT_DATA_	Flat.File.OrderCancellation._TTH_.xls More information: Order Cancellation Template
Flat File Order Adjustments Feed Enumeration value: _POST_FLAT_FILE_PAYMENT_ADJUSTMENT_DATA_	Flat.File.Adjustment._TTH_.xls More information: Adjustments Template
Flat File Order Fulfillment Feed Enumeration value: _POST_FLAT_FILE_FULFILLMENT_DATA_	Flat.File.ShippingConfirm.xls More information: Shipping Confirmation Template
Flat File Sourcing On Demand Feed (Japan only) Enumeration value: _POST_EXPECTED_SHIP_DATE_SOD_FLAT_FILE_	Flat.File.OrderESDSetting.jp.xls More information: Set the estimated ship date for Sourcing on Demand orders using a file

Fulfillment by Amazon feeds

Name	File
FBA Fulfillment Order Feed Enumeration value: _POST_FULFILLMENT_ORDER_REQUEST_DATA_	FulfillmentOrderRequest.xsd More information: XSDs
FBA Fulfillment Order Cancellation Feed Enumeration value: _POST_FULFILLMENT_ORDER_CANCELLATION_REQUEST_DATA_	FulfillmentOrderCancellationRequest.xsd More information: XSDs
FBA Inbound Shipment Carton Information Feed Enumeration value: _POST_FBA_INBOUND_CARTON_CONTENTS_	CartonContentsRequest.xsd More information: Shipping inventory to Amazon's fulfillment network
Flat File FBA Fulfillment Order Feed Enumeration value: _POST_FLAT_FILE_FULFILLMENT_ORDER_REQUEST_DATA_	Flat.File.FulfillmentOrderRequest.xls More information: Upload Multi-Channel Fulfillment File (Create Order)
Flat File FBA Fulfillment Order Cancellation Feed Enumeration value: _POST_FLAT_FILE_FULFILLMENT_ORDER_CANCELLATION_REQUEST_DATA_	Flat.File.FulfillmentOrderCancellationRequest.xls More information: Upload Multi-Channel Fulfillment File (Cancel Order)
Flat File FBA Create Inbound Shipment Plan Feed Enumeration value: _POST_FLAT_FILE_FBA_CREATE_INBOUND_PLAN_	Flat.File.CreateInboundPlanRequest.xls More information: How to Create a Shipping Plan Request
Flat File FBA Update Inbound Shipment Plan Feed Enumeration value: _POST_FLAT_FILE_FBA_UPDATE_INBOUND_PLAN_	Flat.File.UpdateInboundPlanRequest.xls More information: How to Update a Shipping Plan Request
Flat File FBA Create Removal Feed Enumeration value: _POST_FLAT_FILE_FBA_CREATE_REMOVAL_	Flat.File.RemovalOrderRequest.xls More information: Create Removal Order using a Removal Order File

Business feed

Name	File
Flat File Manage Quotes Feed Enumeration value:_RFQ_UPLOAD_FEED_	Amazon Business sellers can use this feed to upload quantity discounts in response to requests from business customers. This functionality is available only in the US, Spain, UK, France, Germany, Italy, India, and Japan marketplaces. For Amazon Business sellers only. More information, see "Manage Quotes feed files" in the Seller Central Help.

Easy Ship feed

Name	File
Easy Ship Feed Enumeration value: _POST_EASYSHIP_DOCUMENTS_	This functionality is available only in the India marketplace. More information: How to get invoice, shipping label, and warranty documents.

Feeds data types

The following data types are used in the Feeds API.

Data type	Description
destination	The location of the feed submission result.
encryptionDetails	The encryption details required for client-side encryption or decryption of the feed content.
FeedOptions	Additional options to control feed submission.
FeedSubmissionInfoList	Information about the feed submission.
SubmitFeedRequest	Request schema for submitting a feed.

destination

The location of the feed submission result.

Name	Description	Required
channel	The distribution channel used to retrieve the feed submission result. Type: string	Yes

Name	Description	Required
url	The URL of the feed submission result. Type: string	Yes

encryptionDetails

The encryption details required for encrypting or decryption the feed data.

Name	Description	Required
standard	The encryption standard used to encrypt or decrypt the feed content. **standard** values: • *AES* – Advanced Encryption Standard Type: string	Yes
initializationVector	The vector used to encrypt or decrypt the feed content. Type: string	Yes
key	The encryption key used to encrypt or decrypt the feed content. Type: string	Yes

FeedOptions

Additional options to control feed submission. This parameter is used for Easy Ship orders.

Name	Description	Required
AmazonOrderId	An Amazon-defined order identifier. Used to identify an Easy Ship order for which you want to get PDF documents. This functionality is available only in the India marketplace. Type: string	No
DocumentType	The type of PDF document that you want to get for the Easy Ship order identified with the AmazonOrderId parameter. This functionality is available only in the India marketplace. Possible values: *ShippingLabel*, *Invoice*, *Warranty*. Type: string	No

feedSubmissionInfoList

Information about the feed submission.

Name	Description	Required
FeedSubmissionId	A unique feed submission identifier. Type: string	Yes
FeedType	The type of feed submitted For **FeedType** values, see FeedType enumeration. Type: string	Yes
SubmittedDate	The date and time when the feed was submitted, in ISO 8601 date time format. Type: string	Yes
FeedProcessingStatus	The processing status of the feed submission. For possible values see FeedProcessingStatus enumeration. Type: string	Yes
StartedProcessingDate	The date and time when the feed processing started, in ISO 8601 date time format. Type: string	No
CompletedProcessingDate	The date and time when the feed processing completed, in ISO 8601 date time format. Type: string	No

SubmitFeedRequest

Request schema for submitting a feed.

Name	Description	Required
MarketplaceIds	A list of one or more marketplace identifiers that you want the feed to be applied to. The feed will be applied to all the marketplaces you specify. When no marketplaces are specified in the feed request, the request returns all marketplaces where the feed request can be submitted. Type: string	No

Name	Description	Required
ContentSha256	The SHA256 hash of the feed content. This value is used to determine if the feed data has been corrupted or tampered with during transit. Use the SHA256 digest stream value that you saved in Task 2. Encrypt and upload the feed data. Type: string	Yes
UploadDestinationId	The upload destination identifier that you saved in Task 1. Create an upload destination. Type: string	Yes
FeedOptions	Additional options to control feed submission. This parameter is used for Easy Ship orders. Type: FeedOptions	No
PurgeAndReplace	A Boolean value that enables the purge and replace functionality. Set to *true* to purge and replace the existing data; otherwise *false*. This value only applies to product-related flat file feed types, which do not have a mechanism for specifying purge and replace in the feed body. Use this parameter only in exceptional cases. Usage is throttled to allow only one purge and replace within a 24-hour period. Type: boolean	No