Use the Wowza Streaming Engine™ media server software Java API to transport data to an Amazon S3 bucket.

Requirements

To use the Wowza Streaming Engine Java `TransportMedia` packages to transport data to Amazon S3, you need:

- Wowza Streaming Engine 4.6.0.03 or later
- Knowledge of programming in Java
- The Wowza IDE
- An Amazon S3 account with an Amazon access key and an Amazon secret key. For more information, see the Amazon documentation About access keys. You'll also need your Amazon S3 bucket name and region name.

Create an Amazon S3 transport provider

The following example creates and initializes a transport provider:
Where:

- **bucketName** is the name of the bucket in Amazon S3.
- **regionName** is the bucket's region. If you don't specify a region, Amazon uses US East as the default.
- **accessKey** and **secretKey** are the access key and secret access key associated with your AWS account.

Amazon S3 transport provider parameters

Static parameters can be used with a transport provider or, in some cases, on a per-transport object basis.
Configure the MediaTransport listeners

The transport model is asynchronous by default, so two listeners are available to determine when object transports are requested and whether they were successful: `IMediaTransportListener` and `IMediaTransportProviderListener`.

`IMediaTransportListener`

The `IMediaTransportListener` provides two callback functions:

```java
public void onTransportProviderAdd (IMediaTransportProvider
```
This example logs the addition or removal of providers:

```java
public class MediaTransportListener implements IMediaTransportListener {
    public void onTransportProviderAdd(IMediaTransportProvider thisProvider) {
        getLogger().info("Adding Provider: "+thisProvider.getTransportProviderName());
    }
    public void onTransportProviderRemove(IMediaTransportProvider thisProvider) {
        getLogger().info("Removing Provider: "+thisProvider.getTransportProviderName());
    }
}
mediaTransport.addTransportListener(new MediaTransportListener());
```

Use this command to add the `IMediaTransportListener` to the transport media system:

```java
mediaTransport.addTransportListener(new MediaTransportListener());
```

`IMediaTransportProviderListener`

The `IMediaTransportProviderListener` provides one callback function:

```java
public void onTransportObjectStatus(int count, IMediaTransportResponse responseObject, String transportProviderName)
```

Each time a transport provider attempts to send an object, the callback function is called to access the status of the transport. In the callback function:

- **count** is the current attempt count for the object being transported. The transport provider will attempt to transport the requested object up to the maximum number of retries set for the object. By default, each object’s maximum retry value is 5.
- **responseObject** is additional information about the transport attempt.
- **transportProviderName** is the name of the transport provider used in the attempt to transport the object.

The following example checks the success of the `responseObject`. If successful, the response...
public class MediaTransportProviderListener implements IMediaTransportProviderListener
{
    public void onTransportObjectStatus(int count, IMediaTransportResponse responseObject, String transportProviderName)
    {
        getLogger().info("onTransportObjectStatus: count: " + count);
        getLogger().info("onTransportObjectStatus: transportProviderName: " + transportProviderName);
        getLogger().info("onTransportObjectStatus: success: " + responseObject.getSuccess());
        getLogger().info("onTransportObjectStatus: basic message: " + responseObject.getBasicMessage());
        byte[] responseCode = responseObject.getSuccessData(IMediaTransportResponse.RESPONSE_INDICATOR);
        if (responseCode != null)
        {
            String thisResponseCode = new String(responseCode);
            getLogger().info("onTransportObjectStatus: Response code: " + thisResponseCode);
        }
        if (responseObject.getSuccess() == true)
        {
            getLogger().info("onTransportObjectStatus: Path of object: " + responseObject.getPath());
            byte[] responseData = responseObject.getSuccessData(IMediaTransportResponse.RESPONSE_DATA);
            if (responseData != null)
            {
                getLogger().info("onTransportObjectStatus: responseData Length: " + responseData.length);
                getLogger().info("onTransportObjectStatus: responseHeaders Length: " + responseHeaders.length);
                getLogger().info("onTransportObjectStatus: responseHeaders :" + new String(responseHeaders));
            }
        }
        else
        {
            byte[] responseData = responseObject.getSuccessData(IMediaTransportResponse.RESPONSE_DATA);
            if (responseData != null)
            {
                getLogger().info("onTransportObjectStatus: responseData Length: " + responseData.length);
                getLogger().info("onTransportObjectStatus: responseData :" + new String(responseData));
            }
        }
    }
}
Use this command to add the `IMediaTransportProviderListener` to the transport provider:

Configure an Amazon S3 bucket

Create a bucket

The following example creates a bucket named `asimpletestbucket` in your account. Note that the XML is placed as the body of the object and the `setPath` is set to a forward slash.

**Note:** This example assumes that the transport provider has been initialized.

Upload content to a bucket

The following example transports an object to the destination path set. Note that Amazon S3
also has options to set storage class and access type for objects.

```java
toolkit.getLogger().info("Filename: \"+", filename, "+ File: FileByte Count: "+ fileBytes.length);  
if ( fileBytes != null )  
    toolkit.getLogger().info("Filename: \"+", filename, "+ UploadFile:  "+ uploadFile);  
    // We do not override anything so this goes into the transport provider configured s
    // settings.  
    MediaTransportDataObject transportDataObject = new MediaTransportDataObject();  
    // Set the path as to the destination of the object  
    transportDataObject.setPath("/"+filename);  
    // Set the content type  
    transportDataObject.setContentType("video/mp4");  
    // Set the data to be transported  
    transportDataObject.setData(fileBytes);  
    // Set the command to use  
    transportDataObject.setCommand("PUT");  
```
Delete a bucket

The following example deletes the bucket named `asimplestbucket` from your account. Note that the `setPath` is set to a forward slash.

**Note:** This example assumes that the transport provider has been initialized.

Configure the Amazon S3 transport object

Transport objects have two parts: options and headers. The options override default values set by the transport provider configuration, and headers provide additional information in the HTTP request.

Transport object options

The following static options are available for transport objects:

- **AMZ_OBJECT_OPTION_BUCKET** – Sets the bucket’s name to something other than the name configured by the transport provider. This is useful when you want objects to go to different buckets.
- **AMZ_OBJECT_OPTION_REGION** – Sets the bucket’s region to something other than the region configured by the transport provider. This is useful when you want objects to
go to different buckets. This is primarily needed to create and delete buckets.

- **AMZ_OBJECT_OPTION_REGION_ENABLED** – Instructs the HTTP request to use the full Amazon S3 hostname rather than just the bucket name. Note that this is required for some older buckets.

The following example sets the **AMZ_OBJECT_OPTION_BUCKET** option; the same syntax can be used to configure the other transport object options.

```java
transportDataObject.setOption(MediaTransportConfigurationAmazonS3.AMZ_OBJECT_OPTION_BUCKET, "anothersimplebucket");
```

**Transport object headers**

Transport object headers enable you to add HTTP headers for specific objects. The following is a list of pre-defined headers:

- **AMZ_OBJECT_HEADER_STORAGE_CLASS**
- **AMZ_OBJECT_HEADER_ACCESS_LIST**
- **AMZ_OBJECT_HEADER_SECURITY_TOKEN**
- **AMZ_OBJECT_HEADER_COPY_SOURCE**
- **AMZ_OBJECT_HEADER_META_DIRECTIVE**
- **AMZ_OBJECT_HEADER_COPY_SOURCE_IF_MATCH**
- **AMZ_OBJECT_HEADER_COPY_SOURCE_IF_NONE_MATCH**
- **AMZ_OBJECT_HEADER_COPY_SOURCE_IF_UNMODIFIED_SINCE**
- **AMZ_OBJECT_HEADER_COPY_SOURCE_IF_MODIFIED_SINCE**
- **AMZ_OBJECT_HEADER_GRANT_READ**
- **AMZ_OBJECT_HEADER_GRANT_WRITE**
- **AMZ_OBJECT_HEADER_GRANT_READ_ACP**
- **AMZ_OBJECT_HEADER_GRANT_WRITE_ACP**
- **AMZ_OBJECT_HEADER_GRANT_FULL_CONTROL**
- **AMZ_OBJECT_HEADER_SERVER_ENCRYPT**
- **AMZ_OBJECT_HEADER_SERVER_ENCRYPT_KEY_ID**
- **AMZ_OBJECT_HEADER_SERVER_ENCRYPT_CONTEXT**

**Note:** You can directly add any HTTP header that you want, and you’re not restricted to the headers listed here. Headers that begin with **x-amz-** will be added correctly to the Amazon S3 signature.

When setting these headers, be sure to provide the correct value. For example, the following command demonstrates how to set the access list header:
transportDataObject.setHeader(MediaTransportConfigurationAmazonS3.AMZ_OBJECT_HEADER_ACCESS_LIST, "public-read");