Object Storage Service

Android SDK Developer Guide

Issue 13

Date 2020-04-27

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 SDK Download Links

Download Address

- Latest version of OBS Android SDK: **OBS Android SDK**
- Earlier versions of OBS Android SDK:
  - **OBS Android SDK 3.20.1**
  - **OBS Android SDK 3.1.3**
  - **OBS Android SDK 3.1.2**
  - **OBS Android SDK 3.0.0**
  - **OBS Android SDK 2.1.7**
  - **OBS Android SDK 2.1.6**

SDK Source Codes and API Documentation

- For details about the SDK source codes, see **GitHub**.
- For the API documentation, see **OBS SDK for Android API**.

Compatibility

- For details about the version revision records, see **ChangeLog**.
- Recommended Android system versions: Android 4.5, 5.0, 5.1, 6.0, 7.0, 8.0, and 8.1 (HTTP/HTTPS)
- Namespace: Compatible with earlier versions (2.1.*x*). All external APIs are contained in the **com.obs.services**, **com.obs.services.model**, and **com.obs.services.exception** packages.
- API functions: Compatible with earlier versions (2.1.*x*).

  NOTE

  If Android 4.4 or an earlier version is used, configure HTTP to access OBS.

ProGuard

```
#okhttp
-dontwarn okhttp3.**
-keep class okhttp3.**{*;}

#okio
```



```
-dontwarn okio.**
-keep class okio.**{*;}

#sdk
-keep class com.obs.services.** {*;}
-keep class com.obs.log.** {*;}

#java-xmlbuilder
-keep class com.jamesmurty.utils.**{*;}
```


2 Example Programs

OBS Android SDK provides abundant example programs for user reference and direct use. These programs can be obtained from the OBS Android SDK development package. For example, files in **eSDK_Storage_OBS_*<versionId>*_Android/samples_android** obtained by decompressing **eSDK_Storage_OBS_*<versionId>*_Android.zip** are example programs. Alternatively, you can click code package names provided in the following table to obtain corresponding example programs.

Example programs include:

Sample Code	Describes
BucketOperationsSample	How to use bucket-related APIs.
ObjectOperationsSample	How to use object-related APIs.
DownloadSample	How to download an object.
CreateFolderSample	How to create a folder.
DeleteObjectsSample	How to delete objects in a batch.
ListObjectsSample	How to list objects.
ListVersionsSample	How to list versioning objects.
ListObjectsInFolderSample	How to list objects in a folder.
ObjectMetaSample	How to customize object metadata.
SimpleMultipartUploadSample	How to perform a multipart upload.
ConcurrentCopyPartSample	How to concurrently copy parts of a large object.
ConcurrentDownloadObjectSample	How to concurrently download parts of a large object.
ConcurrentUploadPartSample	How to concurrently upload parts of a large object.

Sample Code	Describes
RestoreObjectSample	How to download Archive objects.
PostObjectSample	How to perform a browser-based upload.
TemporarySignatureSample	How to use URLs for authorized access.

3 Quick Start

3.1 Before You Start

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

- Ensure that you are familiar with OBS basic concepts, such as **bucket**, **object**, and **AK and SK**.
- You can see **General Examples of ObsClient** to learn how to call OBS Android SDK APIs in a general manner.
- After an API calling is complete using an instance of **ObsClient**, view whether an exception is thrown. If no, the return value is valid. If yes, the operation fails and you need to obtain the error information from an instance of **ObsException**.
- After an API is successfully called by an instance of **ObsClient**, an instance of **HeaderResponse** (or of its sub-class) containing the response headers will be returned.
- Some features are available only in some regions. If the HTTP status code of an API is 405, check whether the region supports this feature.

3.2 Setting Up an OBS Environment

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Step 1 Register a cloud service account.

Before using OBS, ensure that you have a cloud service account.

1. Open a browser.
2. Visit **www.huaweicloud.com/en-us/**.
3. In the upper right corner of the page, click **Register**.
4. Enter the registration information and click **Register**.

Step 2 Subscribe to OBS.

Ensure that your account balance is sufficient before using OBS.

1. Log in to OBS Console.
2. Click **Billing** in the upper right corner of the page to go to the billing center.
3. Click **Top Up**. The dialog box for top-up is displayed.
4. Top up the account as prompted.
5. After the top-up is successful, close the window and go back to the homepage of the management console.
6. Click **Object Storage Service** to subscribe to OBS and enter OBS Console.

Step 3 Create access keys.

OBS uses AKs and SKs in user accounts for signature verification to ensure that only authorized accounts can access specified OBS resources. Detailed explanations about AK and SK are as follows:

- An access key ID (AK) defines a user who accesses the OBS system. An AK belongs to only one user, but one user can have multiple AKs. The OBS system recognizes the users who access the system by their access key IDs.
- An SK is the secret access key on OBS, which is required as the key to access OBS. You can generate authentication information based on SKs and request header fields. An SK matches an AK, and they group into a pair.

The procedure is as follows:

1. Log in to OBS Console.
2. In the upper right corner of the page, hover the cursor over the username and choose **My Credentials**.
3. On the **My Credentials** page, select **Access Keys** in the navigation pane on the left.
4. On the **Access Keys** page, click **Create Access Key**.
5. In the **Create Access Key** dialog box that is displayed, enter the password and verification code.

   NOTE

   - If you have not bound an email address or mobile number, enter only the password.
   - If you have bound an email address and a mobile number, you can select the verification by either email or mobile phone.
6. Click **OK**.
7. In the **Download Access Key** dialog box that is displayed, click **OK** to save the access keys to your browser's default download path.

8. Open the downloaded **credentials.csv** file to obtain the access keys (AK and SK).

NOTE

- Each user can create up to two valid AK/SK pairs.
- To prevent the AK/SK from being leaked, keep them secure. If you click **Cancel** in the dialog box, the access keys will not be downloaded, and cannot be obtained later. You can re-create access keys if you need to use them.

----End

3.3 Preparing a Development Environment

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

- Download Android of the latest version from **Android's official website** and install it.
- After installation, on Android Studio, choose **Settings** > **Appearance & Behavior** > **System Settings** > **Android SDK** to install SDK platforms for supporting application programs, for example, Android 5.0 (Lollipop).

3.4 Installing the SDK

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Import the JAR package in the Android Studio project. The procedure is as follows:

Step 1 Download the **OBS Android SDK development package**.

Step 2 Decompress the development package to obtain all JAR files under the **libs** folder.

Step 3 Open Android Studio. Click **Start a new Android Studio project** to go to the page for creating a project.

Step 4 Fill in the configuration items, such as **Application name** and **Project location**, and finish creating the project as prompted.

Step 5 Find the created project in the file system and copy the JAR files obtained from step 2 to the **app/libs** subfolder of the project.

Step 6 On Android Studio, go to the **app/libs** subfolder of the new project and right-click each **JAR** file in this folder and choose **Add As Library** to add it as a library file of the project.

Step 7 Open the **AndroidManifest.xml** file and add the following code segments to configure the SDK-required permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.GET_TOP_ACTIVITY_INFO" />
```

Step 8 Compile the project and finish OBS SDK integration.

----End

3.5 Obtaining Endpoints

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

- You can click **here** to view the endpoints and regions enabled for OBS.

NOTICE

The SDK allows you to pass endpoints with or without the protocol name. Suppose the endpoint you obtained is **your-endpoint**. The endpoint passed when initializing an instance of **ObsClient** can be **http://your-endpoint**, **https://your-endpoint**, or **your-endpoint**.

3.6 Initializing an Instance of ObsClient

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Each time you want to send an HTTP/HTTPS request to OBS, you must create an instance of **ObsClient**. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
```



```
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Use the instance to access OBS.

// Close obsClient.
obsClient.close();
```

NOTE

For more information, see chapter "Initialization."

3.7 Creating a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

A bucket is a global namespace of OBS and is a data container. It functions as a root directory of a file system and can store objects. The following code shows how to create a bucket:

```
obsClient.createBucket("bucketname");
```

NOTE

- Bucket names are globally unique. Ensure that the bucket you create is named differently from any other bucket.
- A bucket name must comply with the following rules:
  - Contains 3 to 63 characters, chosen from lowercase letters, digits, hyphens (-), and periods (.), and starts with a digit or letter.
  - Cannot be an IP address or similar.
  - Cannot start or end with a hyphen (-) or period (.)
  - Cannot contain two consecutive periods (.), for example, **my..bucket**.
  - Cannot contain periods (.) and hyphens (-) adjacent to each other, for example, **my-.bucket** or **my.-bucket**.
- If you create buckets of the same name, no error will be reported and the bucket properties comply with those set in the first creation request.
- For more information, see **Creating a Bucket**.

NOTICE

- This parameter is not required if the endpoint belongs to the default region (cn-north-1). Otherwise, set this parameter to the region to which the endpoint belongs. Click **here** to query currently valid regions.
- When creating a bucket, you can specify its region. For details, see **Creating a Bucket with Parameters Specified**.

3.8 Uploading an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Sample code:

```
obsClient.putObject("bucketname", "objectname", new ByteArrayInputStream("Hello OBS".getBytes()));
```

NOTE

For more information, see **Object Upload Overview**.

3.9 Downloading an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Sample code:

```
ObsObject obsObject = obsClient.getObject("bucketname", "objectname");
InputStream content = obsObject.getObjectContent();
if (content != null)
{
    BufferedReader reader = new BufferedReader(new InputStreamReader(content));
    while (true)
    {
        String line = reader.readLine();
        if (line == null)
            break;
        Log.i("GetObject", "\n" + line);
    }
    reader.close();
}
```

NOTE

- When you call **ObsClient.getObject**, an instance of **ObsObject** will be returned. This instance contains the content and properties of the object.
- When you call **ObsObject.getObjectContent** to obtain an object input stream, you can read the input stream to obtain its contents. Close the input stream after use.
- For more information, see **Object Download Overview**.

3.10 Listing Objects

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

After objects are uploaded, you may want to view the objects contained in a bucket. Sample code is as follows:

```
ObjectListing objectListing = obsClient.listObjects("bucketname");
for(ObsObject obsObject : objectListing.getObjects()){
    Log.i("ListObjects", " - " + obsObject.getObjectKey() + "  " +  "(size = " +
obsObject.getMetadata().getContentLength() + ")");
}
```

NOTE

- When you call **ObsClient.listObjects**, an instance of **ObjectListing** will be returned. This instance contains the response of the **listObject** request. You can use **ObjetListing.getObjects** to obtain description of all of the listed objects.
- In the previous sample code, 1000 objects will be listed, by default.
- For more information, see **Listing Objects**.

3.11 Deleting an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Sample code:

```
obsClient.deleteObject("bucketname", "objectname");
```

3.12 General Examples of ObsClient

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

After an API calling is complete using an instance of **ObsClient**, view whether an exception is thrown. If no, the return value is valid and an instance of the

HeaderResponse or of its sub-class is returned. If yes, obtain the error information from the instance of **ObsException**.

Sample code:

```
// You can reserve only one global instance of ObsClient in your project.
// ObsClient is thread-safe and can be simultaneously used by multiple threads.
ObsClient obsClient = null;
try
{
    String endPoint = "https://your-endpoint";
    String ak = "*** Provide your Access Key ***";
    String sk = "*** Provide your Secret Key ***";
    // Create an instance of ObsClient.
    obsClient = new ObsClient(ak, sk, endPoint);
    // Invoke the interface to perform operations, for example, upload an object. In the command, localfile indicates the path of the local file to be uploaded. You need to specify the file name.
    HeaderResponse response = obsClient.putObject("bucketname", "objectname", new File("localfile"));
    Log.i("PutObject", response);
}
catch (ObsException e)
{
    Log.e("PutObject", "Response Code: " + e.getResponseCode());
    Log.e("PutObject", "Error Message: " + e.getErrorMessage());
    Log.e("PutObject", "Error Code:       " + e.getErrorCode());
    Log.e("PutObject", "Request ID:      " + e.getErrorRequestId());
    Log.e("PutObject", "Host ID:           " + e.getErrorHostId());
}finally{
    // Close the instance of ObsClient. If this instance is a global one, you do not need to close it every time you complete calling a method.
    // After you call the ObsClient.close method to close an instance of ObsClient, the instance cannot be used any more.
    if(obsClient != null){
        try
        {
            obsClient.close();
        }
        catch (IOException e)
        {
        }
    }
}
```


4 Initialization

4.1 Configuring the AK/SK

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

To use OBS, you need a valid pair of AK and SK for signature authentication. For details, see **Setting Up an OBS Environment**.

After obtaining the AK and SK, you can start initialization according to the following **procedure**.

4.2 Creating an Instance of ObsClient

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

ObsClient functions as the Android client for accessing OBS. It offers users a series of APIs for interaction with OBS and is used for managing and performing operations on resources, such as buckets and objects, stored in OBS. To use OBS Android SDK to send a request to OBS, you need to initialize an instance of ObsClient and modify the default configurations in ObsConfiguration based on actual needs.

- If you use the endpoint to create an instance of ObsClient, all parameters are in their default values and cannot be modified. Sample code is as follows:

  - Sample code for creating an instance of ObsClient using permanent access keys (AK/SK):

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Use the instance to access OBS.

// Close ObsClient.
obsClient.close();
```

  - Sample code for creating an instance of ObsClient using temporary access keys (AK/SK and security token):

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String securityToken = "*** Provide your Security Token ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, securityToken, endPoint);

// Use the instance to access OBS.

// Close ObsClient.
obsClient.close();
```

- If you use ObsConfiguration to create an instance of ObsClient, you can set configuration parameters as needed during the creation. After the instance is created, the parameters cannot be modified. For parameter details, see **Configuring an Instance of ObsClient**. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsConfiguration.
ObsConfiguration config = new ObsConfiguration();
config.setEndPoint(endPoint);
config.setSocketTimeout(30000);
config.setMaxErrorRetry(1);

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, config);

// Use the instance to access OBS.

// Close ObsClient.
obsClient.close();
```

NOTE

- The project can contain one or more instances of ObsClient.
- ObsClient is thread-safe and can be simultaneously used by multiple threads.

NOTICE

After you call **ObsClient.close** to close an instance of ObsClient, the instance cannot be used any more.

4.3 Configuring an Instance of ObsClient

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

When you call the **ObsConfiguration** configuration class to create an instance of **ObsClient**, you can configure the agent, timeout duration, maximum allowed number of connections, and some other parameters listed in the following table.

Parameter	Description	Method	Recommended Value
connectionTimeout	Timeout period for establishing an HTTP/HTTPS connection, in ms. The default value is 60,000.	ObsConfiguration.setConnectionTimeout	[10000, 60000]
socketTimeout	Timeout duration for transmitting data at the Socket layer, in ms. The default value is 60,000.	ObsConfiguration.setSocketTimeout	[10000, 60000]
idleConnectionTime	Allowed connection idle time, in ms. If a connection exceeds the specified value, the connection will be closed. The default value is 30,000.	ObsConfiguration.setIdleConnectionTime	Default value
maxIdleConnections	Maximum number of allowed idle connections in the connection pool. The default value is **1000**.	ObsConfiguration.setMaxIdleConnections	N/A

Parameter	Description	Method	Recommended Value
maxConnections	Maximum number of concurrent HTTP requests. The default value is 1000.	ObsConfiguration.setMaxConnections	Default value
maxErrorRetry	Maximum number of retry attempts (caused by abnormal requests, **500**, **503**, and other errors). The default value is **3**. **NOTE** This parameter is invalid in object upload and download APIs if an interruption occurs after an upload or download task enters the data flow processing phase. In this case, no retry is performed.	ObsConfiguration.setMaxErrorRetry	[0, 5]
endPoint	Endpoint for accessing OBS, which contains the protocol type, domain name (or IP address), and port number. For example, https://your-endpoint:443. For security purposes, you are advised to use HTTPS.	ObsConfiguration.setEndPoint	N/A
httpProxy	HTTP proxy configuration. This parameter is left blank by default.	ObsConfiguration.setHttpProxy	N/A
validateCertificate	Whether to verify the server certificate. The default value is **false**.	ObsConfiguration.setValidateCertificate	N/A

Parameter	Description	Method	Recommended Value
verifyResponseContentType	Whether to verify **ContentType** of the response header. The default value is **true**.	ObsConfiguration.setVerifyResponseContentType	Default value
uploadStreamRetryBufferSize	Size of the cache used for uploading a stream object, in bytes. The default size is 512 KB.	ObsConfiguration.setUploadStreamRetryBuffer-Size	N/A
readBufferSize	Cache size for downloading the object from socket streams, in bytes. Value **-1** indicates that cache is not configured. The default value is **-1**.	ObsConfiguration.setReadBufferSize	N/A
writeBufferSize	Cache size for uploading the object to socket streams, in bytes. Value **-1** indicates that cache is not configured. The default value is **-1**.	ObsConfiguration.setWriteBufferSize	N/A
socketWriteBufferSize	Buffer size for sending a socket, in bytes. This parameter corresponds to **java.net.SocketOptions.SO_SNDBUF**. The default value is **-1**, which indicates no limitation.	ObsConfiguration.setSocketWriteBufferSize	Default value

Parameter	Description	Method	Recommended Value
socketReadBufferSize	Buffer size for receiving a socket, in bytes. This parameter corresponds to **java.net.SocketOptions.SO_RCVBUF**. The default value is **-1**, which indicates no limitation.	ObsConfiguration.setSocketReadBufferSize	Default value
keyManagerFactory	Factory used for generating **javax.net.ssl.KeyManager**. This parameter is left blank by default.	ObsConfiguration.setKeyManagerFactory	N/A
trustManagerFactory	Factory used for generating **javax.net.ssl.TrustManager**. This parameter is left blank by default.	ObsConfiguration.setTrustManagerFactory	N/A
isStrictHostnameVerification	Whether to strictly verify the server-side host name. The default value is **false**.	ObsConfiguration.setIsStrictHostnameVerification	N/A
keepAlive	Whether to use persistent connections to access OBS. The default value is **true**.	ObsConfiguration.setKeepAlive	N/A
cname	Whether to use self-defined domain name to access OBS. The default value is **false**.	ObsConfiguration.setCname	N/A
sslProvider	Provider of **SSLContext**. The **SSLContext** provided by JDK is used by default.	ObsConfiguration.setSslProvider	N/A

Parameter	Description	Method	Recommended Value
httpProtocolType	HTTP protocol type used for accessing OBS servers. The default protocol is HTTP 1.1.	ObsConfiguration.setHttpProtocolType	N/A
httpDispatcher	Customized dispatcher	ObsConfiguration.setHttpDispatcher	N/A

NOTE

- Parameters whose recommended value is **N/A** need to be set according to the actual conditions.
- To improve the upload and download performance of large files in the case that the network bandwidth meets the requirements, you can tune the **socketWriteBufferSize**, **sockeReadBufferSize**, **readBufferSize**, and **writeBufferSize** parameters.
- If the network is unstable, you are advised to set larger values for **connectionTimeout** and **socketTimeout**.
- If the value of **endPoint** does not contain any protocol, HTTPS is used by default.
- For the sake of high DNS resolution performance and OBS reliability, you can set **endPoint** only to the domain name of OBS, instead of the IP address.

4.4 Configuring SDK Logging

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

OBS Android SDK provides the logging function based on the **java.util.logging** library. You can use **LogConfigurator.enableLog** to enable or use **LogConfigurator.disableLog** to disable the logging function. Sample code is as follows:

```
// Set the log level. The default value is LogConfigurator.WARN.
LogConfigurator.setLogLevel(LogConfigurator.INFO);

// Set the number of reserved log files. The default value is 10.
LogConfigurator.setLogFileRolloverCount(5);

// Set the size of each log file, in bytes. The size is not limited by default.
LogConfigurator.setLogFileSize(1024 * 1024 * 10);

// Set the directory for storing log files. By default, the logs are stored in the logs directory of the SD card.
LogConfigurator.setLogFileDir("/storage/sdcard");

// Enable logging.
```



```
LogConfigurator.enableLog();

// Disables logging.
LogConfigurator.disableLog();
```

NOTE

- The logging function is disabled by default. You need to enable it manually.
- For details about SDK logs, see **Log Analysis**.
- If no save directory of log files is configured, log files will be saved in the **logs** directory of the SD card, by default.

4.5 Transparently Transferring the AK and SK

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

OBS Android SDK provides **SecretFlexibleObsClient** that support transparent transfer of AKs and SKs in API functions. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
// Create an instance of ObsConfiguration.
ObsConfiguration config = new ObsConfiguration();
config.setEndPoint(endPoint);

// Create an instance of SecretFlexibleObsClient.
SecretFlexibleObsClient obsClient = new SecretFlexibleObsClient(config);
// Use the instance to access OBS.
String ak1 = "*** Provide your Access Key 1 ***";
String sk1 = "*** Provide your Secret Key 1 ***";
obsClient.listBuckets(ak1, sk1);

String ak2 = "*** Provide your Access Key 2 ***";
String sk2 = "*** Provide your Secret Key 2 ***";
obsClient.listBuckets(ak2, sk2);

// Close obsClient.
obsClient.close();
```

NOTE

SecretFlexibleObsClient is inherited from **ObsClient** and can be used as **ObsClient**.

5 Bucket Management

5.1 Creating a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.createBucket** to create a bucket.

Creating a Bucket Directly

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Create a bucket.
try{
    // The bucket is successfully created.
    HeaderResponse response = obsClient.createBucket("bucketname");
    Log.i("CreateBucket", response.getRequestId());
}
catch (ObsException e)
{
    // Failed to create a bucket.
    Log.e("CreateBucket", "Response Code: " + e.getResponseCode());
    Log.e("CreateBucket", "Error Message: " + e.getErrorMessage());
    Log.e("CreateBucket", "Error Code:       " + e.getErrorCode());
    Log.e("CreateBucket", "Request ID:      " + e.getErrorRequestId());
    Log.e("CreateBucket", "Host ID:           " + e.getErrorHostId());
}
```


NOTE

- Bucket names are globally unique. Ensure that the bucket you create is named differently from any other bucket.
- A bucket name must comply with the following rules:
  - Contains 3 to 63 characters, chosen from lowercase letters, digits, hyphens (-), and periods (.), and starts with a digit or letter.
  - Cannot be an IP address or similar.
  - Cannot start or end with a hyphen (-) or period (.)
  - Cannot contain two consecutive periods (.), for example, **my..bucket**.
  - Cannot contain periods (.) and hyphens (-) adjacent to each other, for example, **my-.bucket** or **my.-bucket**.
- If you create buckets of the same name in a region, no error will be reported and the bucket properties comply with those set in the first creation request.
- The bucket created in the previous example is of the default **ACL** (**private**), in the OBS Standard storage class, and in the default region.

NOTICE

- This parameter is not required if the endpoint belongs to the default region (cn-north-1). If the endpoint belongs to a region other than the default one, set this parameter to the region to which the endpoint belongs. Click **here** to query currently valid regions.
- When creating a bucket, you can specify its region. For details, see **Creating a Bucket with Parameters Specified**.

Creating a Bucket with Parameters Specified

When creating a bucket, you can specify the ACL, storage class, and location for the bucket. OBS provides three storage classes for buckets. For details, see **Storage Class**. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObsBucket obsBucket = new ObsBucket();
obsBucket.setBucketName("bucketname");
// Set the access permission for the bucket to public-read. (The default value is private.)
obsBucket.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ);
// Set the storage class to OBS Archive.

obsBucket.setBucketStorageClass(StorageClassEnum.COLD);
// Set the bucket location.
obsBucket.setLocation("bucketlocation");
// Create a bucket.
try{
    // The bucket is successfully created.
    HeaderResponse response =obsClient.createBucket(obsBucket);
    Log.i("CreateBucket", response.getRequestId());
}
catch (ObsException e)
{
    // Failed to create a bucket.
    Log.e("CreateBucket", "Response Code: " + e.getResponseCode());
```



```
    Log.e("CreateBucket", "Error Message: " + e.getErrorMessage());
    Log.e("CreateBucket", "Error Code:       " + e.getErrorCode());
    Log.e("CreateBucket", "Request ID:      " + e.getErrorRequestId());
    Log.e("CreateBucket", "Host ID:           " + e.getErrorHostId());
}
```

5.2 Listing Buckets

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.listBuckets** to list buckets. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// List bukets.
ListBucketsRequest request = new ListBucketsRequest();
request.setQueryLocation(true)
List<ObsBucket> buckets = obsClient.listBuckets(request);
for(ObsBucket bucket : buckets){
    Log.i("ListBuckets", "BucketName" + bucket.getBucketName());
    Log.i("ListBuckets", "CreationDate" + bucket.getCreationDate());
    Log.i("ListBuckets", "Location:" + bucket.getLocation());
}
```

NOTE

- Obtained bucket names are listed in the lexicographical order.
- Set **ListBucketsRequest.setQueryLocation** to query the bucket location when listing buckets.

5.3 Deleting a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.deleteBucket** to delete a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
```



```
// Delete a bucket.
obsClient.deleteBucket("bucketname");
```

NOTE

- Only empty buckets (without objects and part fragments) can be deleted.
- Bucket deletion is a non-idempotence operation and an error will be reported if the to-be-deleted bucket does not exist.

5.4 Identifying Whether a Bucket Exists

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.headBucket** to identify whether a bucket exists. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

boolean exists = obsClient.headBucket("bucketname");
```

5.5 Obtaining Bucket Metadata

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getBucketMetadata** to obtain the metadata of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketMetadataInfoRequest request = new BucketMetadataInfoRequest("bucketname");
request.setOrigin("http://www.a.com");
//Obtain bucket metadata.
BucketMetadataInfoResult result = obsClient.getBucketMetadata(request);
Log.i("GetBucketMetadata", "\t:" + result.getDefaultStorageClass());
Log.i("GetBucketMetadata", "\t:" + result.getAllowOrigin());
Log.i("GetBucketMetadata", "\t:" + result.getMaxAge());
Log.i("GetBucketMetadata", "\t:" + result.getAllowHeaders());
Log.i("GetBucketMetadata", "\t:" + result.getAllowMethods());
Log.i("GetBucketMetadata", "\t:" + result.getExposeHeaders());
```


NOTE

For details about values of methods, such as **BucketMetadataInfoResult.getAllowMethods**, see the **CORS** configurations of the bucket.

5.6 Managing Bucket ACLs

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

A bucket **ACL** can be configured in three modes:

1. Specify a pre-defined access control policy during bucket creation.
2. Call **ObsClient.setBucketAcl** to specify a pre-defined access control policy.
3. Call **ObsClient.setBucketAcl** to set the ACL directly.

The following table lists the five permission types supported by OBS.

Permission	Description	Value in OBS Android SDK
READ	A grantee with this permission for a bucket can obtain the list of objects in and metadata of the bucket. A grantee with this permission for an object can obtain the object content and metadata.	Permission.PERMISSION_READ
WRITE	A grantee with this permission for a bucket can upload, overwrite, and delete any object in the bucket. This permission is not applicable to objects.	Permission.PERMISSION_WRITE
READ_ACP	A grantee with this permission can obtain the ACL of a bucket or object. A bucket or object owner has this permission permanently.	Permission.PERMISSION_READ_ACP

Permission	Description	Value in OBS Android SDK
WRITE_ACP	A grantee with this permission can update the ACL of a bucket or object. A bucket or object owner has this permission permanently. A grantee with this permission can modify the access control policy and thus the grantee obtains full access permissions.	Permission.PERMISSION_WRITE_ACP
FULL_CONTROL	A grantee with this permission for a bucket has **READ**, **WRITE**, **READ_ACP**, and **WRITE_ACP** permissions for the bucket. A grantee with this permission for an object has **READ**, **WRITE**, **READ_ACP**, and **WRITE_ACP** permissions for the object.	Permission.PERMISSION_FULL_CONTROL

There are five access control policies pre-defined in OBS, as described in the following table:

Policy	Description	Value in OBS Android SDK
private	The owner of a bucket or object has the **FULL_CONTROL** permission for the bucket or object. Other users have no permission to access the bucket or object.	AccessControlList.REST_CANNED_PRIVATE
public-read	If this permission is set for a bucket, everyone can obtain the list of objects, multipart uploads, and object versions in the bucket, as well as metadata of the bucket. If this permission is set for an object, everyone can obtain the content and metadata of the object.	AccessControlList.REST_CANNED_PUBLIC_READ

Policy	Description	Value in OBS Android SDK
public-read-write	If this permission is set for a bucket, everyone can obtain the object list in the bucket, multipart uploads in the bucket, metadata of the bucket; upload objects; delete objects; initialize multipart uploads; upload parts; combine parts; copy parts; and abort multipart uploads. If this permission is set for an object, everyone can obtain the content and metadata of the object.	AccessControlList.REST_CANNED_PUBLIC_READ_WRITE
public-read-delivered	If this permission is set for a bucket, everyone can obtain the object list, multipart uploads, and bucket metadata in the bucket, and obtain the content and metadata of the objects in the bucket. This permission cannot be set for objects.	AccessControlList.REST_CANNED_PUBLIC_READ_DELIVERED
public-read-write-delivered	If this permission is set for a bucket, everyone can obtain the object list in the bucket, multipart uploads in the bucket, metadata of the bucket; upload objects; delete objects; initialize multipart uploads; upload parts; combine parts; copy parts; abort multipart uploads; and obtain content and metadata of objects in the bucket. This permission cannot be set for objects.	AccessControlList.REST_CANNED_PUBLIC_READ_WRITE_DELIVERED

Specifying a Pre-defined Access Control Policy During Bucket Creation

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObsBucket obsBucket = new ObsBucket();
obsBucket.setBucketName("bucketname");
// Set the bucket ACL to public-read-write.
obsBucket.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ_WRITE);
```



```
// Create a bucket.
obsClient.createBucket(obsBucket);
```

Setting a Pre-defined Access Control Policy for a Bucket

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set the bucket ACL to private.
obsClient.setBucketAcl("bucketname", AccessControlList.REST_CANNED_PRIVATE);
```

Directly Setting a Bucket ACL

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
owner.setId("ownerid");
acl.setOwner(owner);
// Grant the FULL_CONTROL permission to a specified user.
acl.grantPermission(new CanonicalGrantee("userid"), Permission.PERMISSION_FULL_CONTROL);
// Grant the READ permission to all users.
acl.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
// Directly set the bucket ACL.
obsClient.setBucketAcl("bucketname", acl);
```

NOTE

The owner or grantee ID needed in the ACL indicates the account ID, which can be viewed on the **My Credential** page of OBS Console.

Obtaining a Bucket ACL

You can call **ObsClient.getBucketAcl** to obtain a bucket ACL. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = obsClient.getBucketAcl("bucketname");
Log.i("GetBucketAcl", acl);
```


5.7 Managing Bucket Policies

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Besides bucket ACLs, bucket owners can use bucket policies to centrally control access to buckets and objects in buckets.

For more information, see **Bucket Policy**.

Setting a Bucket Policy

You can call **ObsClient.setBucketPolicy** to set a bucket policy. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
obsClient.setBucketPolicy("bucketname", "your policy");
```

NOTE

For details about the format (JSON character string) of bucket policies, see the *Object Storage Service API Reference*.

Obtaining a Bucket Policy

You can call **ObsClient.getBucketPolicy** to obtain a bucket policy. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

String policy = obsClient.getBucketPolicy("bucketname");
Log.i("GetBucketPolicy","\t" + policy);
```

Deleting a Bucket Policy

You can call **ObsClient.deleteBucketPolicy** to delete a bucket policy. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.deleteBucketPolicy("bucketname");
```


5.8 Obtaining a Bucket Location

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getBucketLocation** to obtain the location of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

String location = obsClient.getBucketLocation("bucketname");
Log.i("GetBucketLocation", "\t:" + location);
```

NOTE

When creating a bucket, you can specify its location. For details, see **Creating a Bucket**.

5.9 Obtaining Storage Information About a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

The storage information about a bucket includes the used capacity of and the number of objects in the bucket. You can call **ObsClient.getBucketStorageInfo** to obtain the bucket storage information. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketStorageInfo storageInfo = obsClient.getBucketStorageInfo("bucketname");
Log.i("GetBucketStorageInfo", "\t" + storageInfo.getObjectNumber());
Log.i("GetBucketStorageInfo", "\t" + storageInfo.getSize());
```


5.10 Setting or Obtaining a Bucket Quota

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Setting a Bucket Quota

You can call **ObsClient.setBucketQuota** to set the bucket quota. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set the bucket quota to 100 MB.
BucketQuota quota = new BucketQuota(1024 * 1024 * 100);
obsClient.setBucketQuota("bucketname", quota);
```

NOTE

A bucket quota must be a non-negative integer expressed in bytes. The maximum value is 2^{63} - 1.

Obtaining a Bucket Quota

You can call **ObsClient.getBucketQuota** to obtain a bucket quota. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketQuota quota = obsClient.getBucketQuota("bucketname");
Log.i("GetBucketQuota", "\t" + quota.getBucketQuota());
```

5.11 Storage Class

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

OBS allows you to set storage classes for buckets. The storage class of an object defaults to be that of its residing bucket. Different storage classes meet different

needs for storage performance and costs. There are three types of storage class for buckets, as described in the following table:

Storage Class	Description	Value in OBS Android SDK
OBS Standard	Features low access latency and high throughput and is applicable to storing frequently-accessed (multiple times per month) hotspot or small objects (< 1 MB) requiring quick response.	StorageClassEnum.STANDARD
OBS Infrequent Access	Is applicable to storing semi-frequently accessed (less than 12 times a year) data requiring quick response.	StorageClassEnum.WARM
OBS Archive	Is applicable to archiving rarely-accessed (once a year) data.	StorageClassEnum.COLD

For more information, see **Bucket Storage Classes**.

Setting the Storage Class for a Bucket

You can call **ObsClient.setBucketStoragePolicy** to set the storage class for a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set the storage class to OBS Infrequent Access.
BucketStoragePolicyConfiguration storgePolicy = new BucketStoragePolicyConfiguration();
storgePolicy.setBucketStorageClass(StorageClassEnum.WARM);
obsClient.setBucketStoragePolicy("bucketname", storgePolicy);
```

Obtaining the Storage Class of a Bucket

You can call **ObsClient.getBucketStoragePolicy** to obtain the storage class of a bucket. Sample code is as follows:

```
String endPoint = "your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketStoragePolicyConfiguration storagePolicy = obsClient.getBucketStoragePolicy("bucketname");
Log.i("GetBucketStoragePolicy", "\t" + storagePolicy.getBucketStorageClass());
```


6 Object Upload

6.1 Object Upload Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

In OBS, objects are basic data units that users can perform operations on. OBS Android SDK provides abundant APIs for object upload in the following methods:

- **Performing a Streaming Upload**
- **Performing a File-Based Upload**
- **Performing a Multipart Upload**
- **Performing an Appendable Upload**
- **Performing a Resumable Upload**
- **Performing a Browser-Based Upload**

The SDK supports the upload of objects whose size ranges from 0 KB to 5 GB. For streaming upload, appendable upload, and file-based upload, data to be uploaded cannot be larger than 5 GB. If the file is larger than 5 GB, multipart upload (whose part size is smaller than 5 GB) is suitable. Browser-based upload allows files to be uploaded through a browser.

If the uploaded object can be read by anonymous users. After the upload succeeds, anonymous users can access the object data through the object URL. The object URL is in the format of **https://*bucket name.domain name/directory levels/object name***. If the object resides in the root directory of a bucket, its URL does not contain directory levels.

NOTE

In the following codes, "bucketname" indicates the name of the bucket to which the object is to be uploaded. "objectname" is the name of the target object that is expected to be generated in the bucket after the upload. The directory levels can be specified, for example, **src/src1/src2/test.txt**. If no directory level is specified, the object is uploaded to the root directory of the bucket.

6.2 Performing a Streaming Upload

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Streaming upload uses **java.io.InputStream** as the data source of an object. You can call **ObsClient.putObject** to upload the data streams to OBS. Sample code is as follows:

Uploading a Character String (Byte Array)

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

String content = "Hello OBS";
obsClient.putObject("bucketname", "objectname", new ByteArrayInputStream(content.getBytes()));
```

Uploading a Network Stream

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

InputStream inputStream = new URL("http://www.a.com").openStream();
obsClient.putObject("bucketname", "objectname", inputStream);
```

Uploading a File Stream

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

FileInputStream fis = new FileInputStream(new File("localfile")); // localfile indicates the path of the local file to be uploaded. You need to specify the file name.
obsClient.putObject("bucketname", "objectname", fis);
```


NOTE

- To upload a local file, you are advised to use **file-based upload**.
- To upload a large file, you are advised to use **multipart upload**.
- The file to be uploaded cannot exceed 5 GB.

6.3 Performing a File-Based Upload

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

File-based upload uses local files as the data source of objects. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.putObject("bucketname", "objectname", new File("localfile")); // localfile indicates the path of the local file to be uploaded. You need to specify the file name.
```

NOTE

The file to be uploaded cannot exceed 5 GB.

6.4 Obtaining Upload Progresses

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **PutObjectRequest.setProgressListener** to configure the data transmission interface to obtain upload progresses. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

PutObjectRequest request = new PutObjectRequest("bucketname", "objectname");
request.setFile(new File("localfile")); // localfile indicates the path of the local file to be uploaded. You need to specify the file name.
request.setProgressListener(new ProgressListener() {

    @Override
    public void progressChanged(ProgressStatus status) {
```



```
            // Obtain the average upload rate.
            Log.i("PutObject", "AverageSpeed:" + status.getAverageSpeed());
            // Obtain the upload progress in percentage.
            Log.i("PutObject", "TransferPercentage:" + status.getTransferPercentage());
        }
});
// Refresh the upload progress each time 1 MB data is uploaded.
request.setProgressInterval(1024 * 1024L);
obsClient.putObject(request);
```

NOTE

- You can query the upload progress when uploading an object in streaming, file-based, multipart, appendable, or resumable mode.
- If the value of **ProgressStatus.getTransferPercentage()** is **-1**, the content is uploaded in streaming mode. In this case, you must set the object length (**Content-Length**) in the object attribute.

6.5 Creating a Folder

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

There is no folder concept in OBS. All elements in buckets are objects. To create a folder in OBS is essentially to create an object whose size is 0 and whose name ends with a slash (/). Such objects have no difference from other objects and can be downloaded and deleted, except that they are displayed as folders in OBS Console.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

final String keySuffixWithSlash = "parent_directory/";
obsClient.putObject("bucketname", keySuffixWithSlash, new ByteArrayInputStream(new byte[0]));

// In the folder, create an object.
obsClient.putObject("bucketname", keySuffixWithSlash + "objectname", new ByteArrayInputStream("Hello OBS".getBytes()));
```

NOTE

- To create a folder in OBS is to create an object whose size is 0 and whose name ends with a slash (/), in essential.
- To create a multi-level folder, you only need to create the folder with the last level. For example, if you want to create a folder named **src1/src2/src3/**, create it directly, no matter whether the **src1/** and **src1/src2/** folders exist.

6.6 Setting Object Properties

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can set properties for an object when uploading it. Object properties include the object length, MIME type, MD5 value (for verification), and customized metadata. You can set properties for an object that is being uploaded in streaming, file-based, or multipart mode or when **copying the object**.

The following table describes object properties.

Property Name	Description	Default Value
Content-Length	Indicates the object length. If the object length exceeds the flow or file length, the object will be truncated.	Actual length of the stream or file
Content-Type	Indicates the MIME type of the object, which defines the type and network code of the object as well as in which mode and coding will the browser read the object.	application/octet-stream
Content-MD5	Indicates the base64-encoded digest of the object data. It is provided to the OBS server to verify data integrity.	N/A
Storage class	Indicates the storage class of the object. Different storage classes meet different needs for storage performance and costs. The value defaults to be the same as the object's residing bucket and can be changed.	N/A

Property Name	Description	Default Value
Customized metadata	Indicates the user-defined description of the object. It is used to facilitate the customized management on the object.	N/A

Setting the Length for an Object

You can call **ObjectMetadata.setContentLength** to set the length for an object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentLength(1024 * 1024L);//1 MB
obsClient.putObject("bucketname", "objectname", new File("localfile"), metadata);
```

Setting the MIME Type for an Object

You can call **ObjectMetadata.setContentType** to set the MIME type for an object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Upload an image.
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("image/jpeg");
obsClient.putObject("bucketname", "objectname", new File("localimage.jpg"), metadata);
```

NOTE

If this property is not specified, the SDK will automatically identify the MIME type according to the name suffix of the uploaded object. For example, if the name suffix of an object is **.xml** (**.html**), the object will be identified as an application/xml (text/html) file.

Setting the MD5 Value for an Object

You can call **ObjectMetadata.setContentMd5** to set the MD5 value for an object. Sample code is as follows:

```
String endPoint = "https://
your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Upload an image.
ObjectMetadata metadata = new ObjectMetadata();
```



```
metadata.setContentMd5("your md5 which should be encoded by base64");
obsClient.putObject("bucketname", "objectname", new File("localimage.jpg"), metadata);
```

NOTE

- The MD5 value of an object must be a base64-encoded digest.
- The OBS server will compare this MD5 value with the MD5 value obtained by object data calculation. If the two values are not the same, the upload fails with HTTP status code **400** returned.
- If the MD5 value is not specified, the OBS server will skip MD5 value verification.
- You can call **ObsClient.base64Md5** to calculate the **Content-MD5** header directly.

Setting the Storage Class for an Object

You can call **ObjectMetadata.setStorageClass** to set the storage class for an object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectMetadata metadata = new ObjectMetadata();
// Set the storage class to OBS Infrequent Access.
metadata.setObjectStorageClass(StorageClassEnum.WARM);
obsClient.putObject("bucketname", "objectname", new File("localfile"), metadata);
```

NOTE

- If you do not set the storage class for an object, the storage class of the object will be the same as that of its residing bucket.
- OBS provides objects with three storage classes which are consistent with **those** provided for buckets.
- Before downloading an Archive object, you must restore it first.

Customizing Metadata for an Object

You can call **ObjectMetadata.addUserMetadata** to customize metadata for an object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectMetadata metadata = new ObjectMetadata();
metadata.addUserMetadata("property1", "property-value1");
metadata.getMetadata().put("property2", "property-value2");
obsClient.putObject("bucketname", "objectname", new File("localfile"), metadata);
```


NOTE

- In the preceding code, two pieces of metadata named **property1** and **property2** are customized and their respective values are set to **property-value1** and **property-value2**.
- An object can have multiple pieces of metadata whose size cannot exceed 8 KB.
- The customized object metadata can be obtained by using **ObsClient.getObjectMetadata**. For details, see **Obtaining Object Properties**.
- When you call **ObsClient.getObject** to download an object, its customized metadata will also be downloaded.

6.7 Performing a Multipart Upload

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

To upload a large file, multipart upload is recommended. Multipart upload is applicable to many scenarios, including:

- Files to be uploaded are larger than 100 MB.
- The network condition is poor. Connection to the OBS server is constantly down.
- Sizes of files to be uploaded are uncertain.

Multipart upload consists of three phases:

Step 1 Initialize a multipart upload (**ObsClient.initiateMultipartUpload**).

Step 2 Upload parts one by one or concurrently (**ObsClient.uploadPart**).

Step 3 Combine parts (**ObsClient.completeMultipartUpload**) or abort the multipart upload (**ObsClient.abortMultipartUpload**).

----End

Initializing a Multipart Upload

Before upload, you need to notify OBS of initializing a multipart upload. This operation will return an upload ID (globally unique identifier) created by the OBS server to identify the multipart upload. You can use this upload ID to initiate related operations, such as aborting a multipart upload, listing multipart uploads, and listing uploaded parts.

You can call **ObsClient.initiateMultipartUpload** to initialize a multipart upload.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
```



```
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest("bucketname",
"objectname");
ObjectMetadata metadata = new ObjectMetadata();
metadata.addUserMetadata("property", "property-value");
metadata.setContentType("text/plain");
request.setMetadata(metadata);
InitiateMultipartUploadResult result = obsClient.initiateMultipartUpload(request);

String uploadId = result.getUploadId();
Log.i("InitiateMultipartUpload", "\t" + uploadId);
```

NOTE

- Call **InitiateMultipartUploadRequest** to specify the name and owning bucket of the uploaded object.
- In **InitiateMultipartUploadRequest**, you can specify the MIME type, storage class, and customized metadata for the object.
- The upload ID of the multipart upload returned by **InitiateMultipartUploadResult.getUploadId** will be used in follow-up operations.

Uploading a Part

After initializing a multipart upload, you can specify the object name and upload ID to upload a part. Each upload part has a part number (ranging from 1 to 10000). For parts with the same upload ID, their part numbers are unique and identify their comparative locations in the object. If you use the same part number to upload two parts, the latter one being uploaded will overwrite the former. Except for the part last uploaded whose size ranges from 0 to 5 GB, sizes of the other parts range from 100 KB to 5 GB. Parts are uploaded in random order and can be uploaded through different processes or machines. OBS will combine them into the object based on their part numbers.

You can call **ObsClient.uploadPart** to upload a part.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

List<PartEtag> partEtags = new ArrayList<PartEtag>();
// Upload the first part.
UploadPartRequest request = new UploadPartRequest("bucketname", "objectname");
// Set the upload ID.
request.setUploadId(uploadId);
// Set the part number, which ranges from 1 to 10000.
request.setPartNumber(1);
// Set the large file to be uploaded. localfile is the path of the local file to be uploaded. You need to specify the file name.
request.setFile(new File("localfile"));

// Set the part size.
request.setPartSize(5 * 1024 * 1024L);
UploadPartResult result = obsClient.uploadPart(request);
partEtags.add(new PartEtag(result.getEtag(), result.getPartNumber()));

// Upload the second part.
request = new UploadPartRequest("bucketname", "objectname");
// Set the upload ID.
request.setUploadId(uploadId);
// Set the part number.
request.setPartNumber(2);
// Set the large file to be uploaded.
```



```
request.setFile(new File("localfile"));
// Set the offset for the second part.
request.setOffset(5 * 1024 * 1024L);
// Set the part size.
request.setPartSize(5 * 1024 * 1024L);
result = obsClient.uploadPart(request);
partEtags.add(new PartEtag(result.getEtag(), result.getPartNumber()));
```

NOTE

- Except the part last uploaded, other parts must be larger than 100 KB. Part sizes will not be verified during upload because which one is last uploaded is not identified until parts are combined.
- OBS will return ETags (MD5 values) of the received parts to users.
- To ensure data integrity, set **UploadPartRequest.setAttachMd5** to **true** to make the SDK automatically calculate the MD5 value (valid only when the data source is a local file) of each part and add the MD5 value to the **Content-MD5** request header. The OBS server will compare the MD5 value contained by each part and that calculated by the SDK to verify the data integrity.
- You can call **UploadPartRequest.setContentMd5** to set the MD5 value of the uploaded data directly. If this value is set, the **UploadPartRequest.setAttachMd5** parameter becomes ineffective.
- Part numbers range from 1 to 10000. If a part number exceeds this range, OBS will return **a 400 Bad Request** error.

Combining Parts

After all parts are uploaded, call the API for combining parts to generate the object. Before this operation, valid part numbers and ETags of all parts must be sent to OBS. After receiving this information, OBS verifies the validity of each part one by one. After all parts pass the verification, OBS combines these parts to form the final object.

You can call **ObsClient.completeMultipartUpload** to combine parts.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

List<PartEtag> partEtags = new ArrayList<PartEtag>();

// First part
PartEtag part1 = new PartEtag();
part1.setPartNumber(1);
part1.seteTag("etag1");
partEtags.add(part1);

// Second part
PartEtag part2 = new PartEtag();
part2.setPartNumber(2);
part2.seteTag("etag2");
partEtags.add(part2);

CompleteMultipartUploadRequest request = new CompleteMultipartUploadRequest("bucketname",
"objectname", uploadId, partEtags);

obsClient.completeMultipartUpload(request);
```


NOTE

- In the preceding code, **partEtags** indicates the list of part numbers and ETags of uploaded parts. These parts are listed in ascending order by part number.
- Part numbers can be inconsecutive.

Concurrently Uploading Parts

Multipart upload is mainly used for large file upload or when the network condition is poor. The following sample code shows how to concurrently upload parts involved in a multipart upload:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
final String bucketName = "bucketname";
final String objectKey = "objectname";
// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Initialize the thread pool.
ExecutorService executorService = Executors.newFixedThreadPool(20);
final File largeFile = new File("localfile");

// Initialize the multipart upload.
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucketName, objectKey);
InitiateMultipartUploadResult result = obsClient.initiateMultipartUpload(request);

final String uploadId = result.getUploadId();
Log.i("UploadPart", "\t"+ uploadId + "\n");

// Set the part size to 100 MB.
long partSize = 100 * 1024 * 1024L;
long fileSize = largeFile.length();

// Calculate the number of parts to be uploaded.
long partCount = fileSize % partSize == 0 ? fileSize / partSize : fileSize / partSize + 1;

final List<PartEtag> partEtags = Collections.synchronizedList(new ArrayList<PartEtag>());

// Concurrently upload parts.
for (int i = 0; i < partCount; i++)
{
    // Start position of a part in the file
    final long offset = i * partSize;
    // Part size
    final long currPartSize = (i + 1 == partCount) ? fileSize - offset : partSize;
    // Part number
    final int partNumber = i + 1;
    executorService.execute(new Runnable()
    {
        @Override
        public void run()
        {
            UploadPartRequest uploadPartRequest = new UploadPartRequest();
            uploadPartRequest.setBucketName(bucketName);
            uploadPartRequest.setObjectKey(objectKey);
            uploadPartRequest.setUploadId(uploadId);
            uploadPartRequest.setFile(largeFile);
            uploadPartRequest.setPartSize(currPartSize);
            uploadPartRequest.setOffset(offset);
            uploadPartRequest.setPartNumber(partNumber);

            UploadPartResult uploadPartResult;
            try
            {
                uploadPartResult = obsClient.uploadPart(uploadPartRequest);
```



```
                Log.i("UploadPart", "Part#" + partNumber + " done\n");
                partEtags.add(new PartEtag(uploadPartResult.getEtag(), uploadPartResult.getPartNumber()));
            }
            catch (ObsException e)
            {
                Log.e("UploadPart", e.getMessage(), e);
            }
        }
    });
}

// Wait until the upload is complete.
executorService.shutdown();
while (!executorService.isTerminated())
{
    try
    {
        executorService.awaitTermination(5, TimeUnit.SECONDS);
    }
    catch (InterruptedException e)
    {
        Log.e("UploadPart", e.getMessage(), e);
    }
}

// Combine parts.
CompleteMultipartUploadRequest completeMultipartUploadRequest =
    new CompleteMultipartUploadRequest(bucketName, objectKey, uploadId, partEtags);
obsClient.completeMultipartUpload(completeMultipartUploadRequest);
```

NOTE

When uploading a large file, use **UploadPartRequest.setOffset** and **UploadPartRequest.setPartSize** to determine the start and end positions of each part.

Aborting a Multipart Upload

After a multipart upload is aborted, you cannot use its upload ID to perform any operation and the uploaded parts will be deleted by OBS.

When an object is being uploaded in multi-part mode or an object fails to be uploaded, parts are generated in the bucket. These parts occupy your storage space. You can cancel the multi-part uploading task to delete unnecessary parts, thereby saving the storage space.

You can call **ObsClient.abortMultipartUpload** to abort a multipart upload.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AbortMultipartUploadRequest request = new AbortMultipartUploadRequest("bucketname", "objectname",
uploadId);

obsClient.abortMultipartUpload(request);
```

Listing Uploaded Parts

You can call **ObsClient.listParts** to list successfully uploaded parts of a multipart upload.

The following table describes the parameters involved in this API.

Parameter	Description	Method in OBS Android SDK
bucketName	Bucket name	ListPartsRequest.setBucketName
key	Object name	ListPartsRequest.setKey
uploadId	Upload ID, which globally identifies a multipart upload. The value is in the returned result of **ObsClient.initiateMultipartUpload**.	ListPartsRequest.setUploadId
maxParts	Maximum number of parts that can be listed per page.	ListPartsRequest.setMaxParts
partNumberMarker	Part number after which listing parts begins. Only parts whose part numbers are larger than this value will be listed.	ListPartsRequest.setPartNumberMarker

- Listing parts in simple mode

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

//List the uploaded parts. uploadId is obtained from initiateMultipartUpload.
ListPartsRequest request = new ListPartsRequest("bucketname", "objectname");
request.setUploadId(uploadId);
ListPartsResult result = obsClient.listParts(request);

for(Multipart part : result.getMultipartList()){
// Part number, specified when being uploaded
   Log.i("ListParts", "\t"+part.getPartNumber());
// Part size
   Log.i("ListParts","\t"+part.getSize());
// Part ETag
   Log.i("ListParts","\t"+part.getEtag());
// Time when the part was last uploaded
   Log.i("ListParts","\t"+part.getLastModified());
}
```

NOTE

- Information about a maximum of 1000 parts can be listed each time. If an upload of the specific upload ID contains more than 1000 parts and **ListPartsResult.isTruncated** is **true** in the returned result, not all parts are listed. In such cases, you can use **ListPartsResult.getNextPartNumberMarker** to obtain the start position for next listing.
- If you want to obtain all parts involved in a specific upload ID, you can use the paging mode for listing.

- Listing all parts

If the number of parts of a multipart upload is larger than 1000, you can use the following sample code to list all parts.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

//List the uploaded parts. uploadId is obtained from initiateMultipartUpload.
ListPartsRequest request = new ListPartsRequest("bucketname", "objectname");
request.setUploadId(uploadId);
ListPartsResult result;

do{
    result = obsClient.listParts(request);
    for(Multipart part : result.getMultipartList()){
// Part number, specified when being uploaded
        Log.i("ListParts","\t"+part.getPartNumber());
// Part size
        Log.i("ListParts","\t"+part.getSize());
// Part ETag
        Log.i("ListParts","\t"+part.getEtag());
// Time when the part was last uploaded
        Log.i("ListParts","\t"+part.getLastModified());
    }
    request.setPartNumberMarker(Integer.parseInt(result.getNextPartNumberMarker()));
}while(result.isTruncated());
```

Listing Multipart Uploads

You can call **ObsClient.listMultipartUploads** to list multipart uploads. The following table describes parameters involved in **ObsClient.listMultipartUploads**.

Parameter	Description	Method in OBS Android SDK
bucketName	Bucket name	ListMultipartUploadsRequest.setBucketName
prefix	Prefix that the object names in the multipart uploads to be listed must contain	ListMultipartUploadsRequest.setPrefix
delimiter	Character used to group object names involved in multipart uploads. If the object name contains the **delimiter** parameter, the character string from the first character to the first delimiter in the object name is grouped under a single result element, **commonPrefix**. (If a prefix is specified in the request, the prefix must be removed from the object name.)	ListMultipartUploadsRequest.setDelimiter

Parameter	Description	Method in OBS Android SDK
maxUploads	Maximum number of listed multipart uploads. The value ranges from 1 to 1000. If the value is not in this range, 1000 uploads are listed by default.	ListMultipartUploadsRequest.setMaxUploads
keyMarker	Object name to start with when listing multipart uploads	ListMultipartUploadsRequest.setKeyMarker
uploadIdMarker	Upload ID after which the multipart upload listing begins. It is effective only when used with **keyMarker** so that multipart uploads after **uploadIdMarker** of **keyMarker** will be listed.	ListMultipartUploadsRequest.setUploadIdMarker

- Listing multipart uploads in simple mode

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListMultipartUploadsRequest request = new ListMultipartUploadsRequest("bucketname");

MultipartUploadListing result = obsClient.listMultipartUploads(request);
for(MultipartUpload upload : result.getMultipartTaskList()){
    Log.i("ListMultipartUploads","\t" + upload.getUploadId());
    Log.i("ListMultipartUploads","\t" + upload.getObjectKey());
    Log.i("ListMultipartUploads","\t" + upload.getInitiatedDate());
}
```

NOTE

- Information about a maximum of 1000 multipart uploads can be listed each time. If a bucket contains more than 1000 multipart uploads and **MultipartUploadListing.isTruncated** is **true**, not all uploads are listed. In such cases, you can use **MultipartUploadListing.getNextKeyMarker** and **MultipartUploadListing.getNextUploadIdMarker** to obtain the start position for next listing.
- If you want to obtain all multipart uploads in a bucket, you can list them in paging mode.

- Listing all multipart uploads in paging mode

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
String uploadId = "upload id from initiateMultipartUpload";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListMultipartUploadsRequest request = new ListMultipartUploadsRequest("bucketname");
MultipartUploadListing result;

do{
    result = obsClient.listMultipartUploads(request);
```



```
    for(MultipartUpload upload : result.getMultipartTaskList()){
        Log.i("ListMultipartUploads","\t" + upload.getUploadId());
        Log.i("ListMultipartUploads","\t" + upload.getObjectKey());
        Log.i("ListMultipartUploads","\t" + upload.getInitiatedDate());
    }
    request.setKeyMarker(result.getNextKeyMarker());
    request.setUploadIdMarker(result.getNextUploadIdMarker());
}while(result.isTruncated());
```

6.8 Configuring Lifecycle Management

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

When uploading an object or initializing a multipart upload, you can directly set the expiration time for the object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

PutObjectRequest request = new PutObjectRequest ("bucketname", "objectname");
request.setFile(new File("localfile"));
// When uploading an object, set the object to expire after 30 days.
request.setExpires(30);
obsClient.putObject(request);


InitiateMultipartUploadRequest request2 = new InitiateMultipartUploadRequest("bucketname",
"objectname");
// When initializing a multipart upload, set the object to expire 60 days after combination.
request2.setExpires(60);
obsClient.initiateMultipartUpload(request);
```

NOTE

- This mode specifies the time duration in days after which an object will expire. The OBS server automatically clears expired objects.
- The object expiration time set in this mode takes precedence over the bucket lifecycle rule.

6.9 Performing an Appendable Upload

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Appendable upload allows you to upload an object in appendable mode and then append data to the object. You can call **ObsClient.appendObject** to perform an appendable upload. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Upload an object in appendable mode.
AppendObjectRequest request = new AppendObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("objectname");
request.setPosition(0L);
request.setInput(new ByteArrayInputStream("Hello OBS".getBytes()));
AppendObjectResult result = obsClient.appendObject(request);

// Append data to the object.
request.setPosition(result.getNextPosition());
request.setInput(new ByteArrayInputStream("Hello OBS Again".getBytes()));
result = obsClient.appendObject(request);

Log.i("AppendObject", "NextPosition:" + result.getNextPosition());
Log.i("AppendObject",  "Etag:" + result.getEtag());
// Use the API for obtaining object properties to get the start position for next appending.
ObjectMetadata metadata = obsClient.getObjectMetadata("bucketname", "objectname");
Log.i("AppendObject", "NextPosition from metadata:" + metadata.getNextPosition());
```

NOTE

- Objects uploaded using **ObsClient.putObject**, referred to as normal objects, can overwrite objects uploaded using **ObsClient.appendObject**, referred to as appendable objects. Data cannot be appended to an appendable object anymore once the object has been overwritten by a normal object.
- When you upload an object for the first time in appendable mode, an exception will be thrown (status code **409**) if a normal object with the same name exists.
- The ETag returned for an appendable upload is the ETag for the uploaded content, rather than that of the whole object.
- Data size in each appendable upload cannot exceed 5 GB, and 10,000 times of appendable uploads can be performed on a single object.
- After an appendable upload is successful, you can use **AppendObjectResult.getNextPosition** or call **ObsClient.getObjectMetadata** to get the start position for next appending.

6.10 Performing a Multipart Copy

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

As a special case of multipart upload, multipart copy implements multipart upload by copying the whole or partial object in a bucket. You can call **ObsClient.copyPart** to copy parts. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
```



```
String sk = "*** Provide your Secret Key ***";

final String destBucketName = "destbucketname";
final String destObjectKey = "destobjectname";
final String sourceBucketName = "sourcebucketname";
final String sourceObjectKey = "sourceobjectname";
// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Initialize the thread pool.
ExecutorService executorService = Executors.newFixedThreadPool(20);

// Initialize the multipart upload.
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(destBucketName,
destObjectKey);
InitiateMultipartUploadResult result = obsClient.initiateMultipartUpload(request);

final String uploadId = result.getUploadId();
Log.i("CopyPart","\t"+ uploadId + "\n");

// Obtain information about the large object.
ObjectMetadata metadata = obsClient.getObjectMetadata(sourceBucketName, sourceObjectKey);
// Set the part size for copying to 100 MB.
long partSize = 100 * 1024 * 1024L;
long objectSize = metadata.getContentLength();

// Calculate the number of parts to be copied.
long partCount = objectSize % partSize == 0 ? objectSize / partSize : objectSize / partSize + 1;

final List<PartEtag> partEtags = Collections.synchronizedList(new ArrayList<PartEtag>());

// Concurrently copy parts.
for (int i = 0; i < partCount; i++)
{
// Start position for copying
   final long rangeStart = i * partSize;
// End position for copying
   final long rangeEnd = (i + 1 == partCount) ? objectSize - 1 : rangeStart + partSize - 1;
// Part number
   final int partNumber = i + 1;
   executorService.execute(new Runnable()
   {

      @Override
      public void run()
      {
         CopyPartRequest request = new CopyPartRequest();
         request.setUploadId(uploadId);
         request.setSourceBucketName(sourceBucketName);
         request.setSourceObjectKey(sourceObjectKey);
         request.setDestinationBucketName(destBucketName);
         request.setDestinationObjectKey(destObjectKey);
         request.setByteRangeStart(rangeStart);
         request.setByteRangeEnd(rangeEnd);
         request.setPartNumber(partNumber);
         CopyPartResult result;
         try
         {
            result = obsClient.copyPart(request);
            Log.i("CopyPart","Part#" + partNumber + " done\n");
            partEtags.add(new PartEtag(result.getEtag(), result.getPartNumber()));
         }
         catch (ObsException e)
         {
            Log.e("CopyPart", e.getMessage(), e);
         }
      }
   });
}
```



```
// Wait until the copy is complete.
executorService.shutdown();
while (!executorService.isTerminated())
{
    try
    {
        executorService.awaitTermination(5, TimeUnit.SECONDS);
    }
    catch (InterruptedException e)
    {
        Log.e("CopyPart", e.getMessage(), e);
    }
}

// Combine parts.
CompleteMultipartUploadRequest completeMultipartUploadRequest =
    new CompleteMultipartUploadRequest(destBucketName, destObjectKey, uploadId, partEtags);
obsClient.completeMultipartUpload(completeMultipartUploadRequest);
```

6.11 Performing a Resumable Upload

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Uploading large files often fails due to poor network conditions or program breakdowns. It is a waste of resources to restart the upload process upon an upload failure, and the restarted upload process may still suffer from the unstable network. To resolve such issues, you can use the API for resumable upload, whose working principle is to divide the to-be-uploaded file into multiple parts and upload them separately. The upload result of each part is recorded in a checkpoint file in real time. Only when all parts are successfully uploaded, the result indicating a successful upload will be returned. Otherwise, an exception is thrown to remind you of calling the API again for re-uploading. Based on the upload status of each part recorded in the checkpoint file, the re-uploading will upload the parts failed to be uploaded previously, instead of uploading all parts. By virtue of this, resources are saved and efficiency is improved.

You can call **ObsClient.uploadFile** to perform a resumable upload. The following table describes the parameters involved in this API.

Parameter	Description	Method in OBS Android SDK
bucketName	(Mandatory) Bucket name	UploadFileRequest.setBucketName
objectKey	(Mandatory) Object name	UploadFileRequest.setObjectKey
uploadFile	(Mandatory) Local file to be uploaded	UploadFileRequest.setUploadFile

Parameter	Description	Method in OBS Android SDK
partSize	Part size, in bytes. The value ranges from 100 KB to 5 GB and defaults to 5 MB.	UploadFileRequest.setPartSize
taskNum	Maximum number of threads that can be concurrently executed for uploading. The default value is **1**.	UploadFileRequest.setTaskNum
enableCheckpoint	Whether to enable the resumable upload mode. The default value is **false**, which indicates that this mode is disabled.	UploadFileRequest.setEnableCheckpoint
checkpointFile	File used to record the upload progress. This parameter is effective only in the resumable upload mode. If this parameter is null, the file will be in the same directory as the local file to be uploaded.	UploadFileRequest.setCheckpointFile
objectMetadata	Object properties	UploadFileRequest.setObjectMetadata
enableCheckSum	Whether to verify the content of the to-be-uploaded file. This parameter is effective only in the resumable upload mode. The default value is **false,** which indicates that the content will not be verified.	UploadFileRequest.setEnableCheckSum
progressListener	Configure the data transmission listener to obtain upload progresses.	UploadFileRequest.setProgressListener

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
```



```
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

UploadFileRequest request = new UploadFileRequest("bucketname", "objectname");
// Set the large file to be uploaded. localfile is the path of the local file to be uploaded. You need to specify the file name.
request.setUploadFile("localfile");
// Set the maximum number of threads that can be concurrently uploaded.
request.setTaskNum(5);
// Set the part size to 10 MB.
request.setPartSize(10 * 1024 * 1024);
// Enable resumable upload.
request.setEnableCheckpoint(true);
try{
    // Perform a resumable upload.
   CompleteMultipartUploadResult result = obsClient.uploadFile(request);
}catch (ObsException e) {
    // When an exception occurs, you can call the API for resumable upload again to perform re-uploading.
}
```

NOTE

- The API for resumable upload, which is implemented based on **multipart upload**, is an encapsulated and enhanced version of multipart upload.
- This API saves resources and improves efficiency upon the re-upload, and speeds up the upload process by concurrently uploading parts. Because this API is transparent to users, users are free from concerns about internal service details, such as the creation and deletion of checkpoint files, division of objects, and concurrent upload of parts.
- The default value of the **enableCheckpoint** parameter is **false**, which indicates that the resumable upload mode is disabled. In such cases, this API degrades to the simple encapsulation of multipart upload, and no checkpoint file will be generated.
- **checkpointFile** and **enableCheckSum** are effective only when **enableCheckpoint** is **true**.

6.12 Performing a Browser-Based Upload

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Performing a browser-based upload is to upload objects to a specified bucket in HTML form. The maximum size of an object is 5 GB.

You can call **ObsClient.createPostSignature** to generate request parameters for browser-based upload. You can use code to simulate a browser-based upload. For details, see **PostObjectSample**. You can also perform a browser-based upload.

Step 1 Call **ObsClient.createPostSignature** to generate request parameters for authentication.

Step 2 Prepare an HTML form page.

Step 3 Enter the request parameters in the HTML page.

Step 4 Select a local file and upload it in browser-based mode.

----End

NOTE

There are two request parameters generated:

- **policy**, which corresponds to the **policy** field in the form
- **signature**: corresponds to the **signature** field in the form.

The following sample code shows how to generate the parameters in a browser-based upload request.

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

PostSignatureRequest request = new PostSignatureRequest();
// Fill in parameters in the form.
Map<String, Object> formParams = new HashMap<String, Object>();
// Set the object ACL to public-read.
formParams.put("x-obs-acl", "public-read");
// Set the MIME type for the object.
formParams.put("content-type", "text/plain");


request.setFormParams(formParams);
// Set the validity period for the browser-based upload request, in seconds.
request.setExpires(3600);
PostSignatureResponse response = obsClient.createPostSignature(request);

// Obtain the request parameters.
Log.i("CreatePostSignature", "\t" + response.getPolicy());
Log.i("CreatePostSignature", "\t" + response.getSignature());
```

Code of an HTML form example is as follows:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>

<form action="http://bucketname.your-endpoint/" method="post" enctype="multipart/form-data">
Object key
<!-- Object name -->
<input type="text" name="key" value="objectname" />
<p>
ACL
<!-- Object ACL -->
<input type="text" name="x-obs-acl" value="public-read" />
<p>
Content-Type
<!-- Object MIME type -->
<input type="text" name="content-type" value="text/plain" />
<p>
<!-- Base64 code of the policy -->
<input type="hidden" name="policy" value="*** Provide your policy ***" />
<!-- AK -->
<input type="hidden" name="AccessKeyId" value="*** Provide your access key ***"/>
<!-- Signature information -->
<input type="hidden" name="signature" value="*** Provide your signature ***"/>
```



```
<input name="file" type="file" />
<input name="submit" value="Upload" type="submit" />
</form>
</body>
</html>
```

NOTE

- Values of **policy** and **signature** in the HTML form are obtained from the returned result of **ObsClient.createPostSignature**.
- You can directly download the HTML form example: **PostDemo**.

7 Object Download

7.1 Object Download Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

OBS Android SDK provides abundant APIs for object download in the following modes:

- **Performing a Streaming Download**
- **Performing a Partial Download**
- **Performing a Resumable Download**

You can call **ObsClient.getObject** to download an object.

7.2 Performing a Streaming Download

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);
```



```
ObsObject obsObject = obsClient.getObject("bucketname", "objectname");

// Obtain the object content.
Log.i("GetObject", "Object content:");
InputStream input = obsObject.getObjectContent();
byte[] b = new byte[1024];
ByteArrayOutputStream bos = new ByteArrayOutputStream();
int len;
while ((len=input.read(b)) != -1){
    bos.write(b, 0, len);
}

Log.i("GetObject", new String(bos.toByteArray()));
bos.close();
input.close();
```

NOTE

- After **ObsClient.getObject** is called, an instance of **ObsObject** will be returned. This instance contains the residing bucket, name, properties, and input streams of the object.
- You can perform operations on the input streams of an object to read and write the object contents to a local file or to the memory.

NOTICE

Object input streams obtained by **ObsObject.getObjectContent** must be closed explicitly. Otherwise, resource leakage occurs.

7.3 Performing a Partial Download

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

When only partial data of an object is required, you can download data falling within a specific range. If the specified range is 0 to 1000, data at the 0th to the 1000th bytes, 1001 bytes in total, will be returned. If the specified range is invalid, data of the whole object will be returned. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectname");
// Set the start position and end position for downloading.
request.setRangeStart(0l);
request.setRangeEnd(1000l);
ObsObject obsObject = obsClient.getObject(request);

// Obtain data.
byte[] buf = new byte[1024];
InputStream in = obsObject.getObjectContent();
```



```
for (int n = 0; n != -1; ) {
    n = in.read(buf, 0, buf.length);
}

in.close();
```

NOTE

- If the specified range is invalid (because the start or end position is set to a negative integer or the range is larger than the object length), data of the whole object will be returned.
- This download method also can be used to concurrently download parts of a large object. For details about the sample code, see **ConcurrentDownloadObjectSample**.

7.4 Obtaining Download Progresses

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **GetObjectRequest.setProgressInterval** to configure the data transmission interface to obtain download progresses. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectname");
request.setProgressListener(new ProgressListener() {

    @Override
    public void progressChanged(ProgressStatus status) {
        // Obtain the average download rate.
        Log.i("GetObject", "AverageSpeed:" + status.getAverageSpeed());
        // Obtain the download progress in percentage.
        Log.i("GetObject", "TransferPercentage:" + status.getTransferPercentage());
    }
});
// Refresh the upload progress each time 1 MB data is uploaded.
request.setProgressInterval(1024 * 1024L);
ObsObject obsObject = obsClient.getObject(request);

// Obtain the object content.
System.out.println("Object content:");
InputStream input = obsObject.getObjectContent();
byte[] b = new byte[1024];
ByteArrayOutputStream bos = new ByteArrayOutputStream();
int len;
while ((len=input.read(b)) != -1){
    bos.write(b, 0, len);
}

System.out.println(new String(bos.toByteArray()));
bos.close();
input.close();
```


NOTE

You can obtain the download progress when downloading an object in streaming, partial, or resumable mode.

7.5 Performing a Conditioned Download

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

When downloading an object, you can specify one or more conditions. Only when the conditions are met, the object will be downloaded. Otherwise, an exception will be thrown and the download will fail.

You can set the following conditions:

Parameter	Description	Method in OBS Android SDK
If-Modified-Since	Returns the object if it is modified after the time specified by this parameter; otherwise, an exception is thrown.	GetObjectRequest.setIfModifiedSince
If-Unmodified-Since	Returns the object if it remains unchanged since the time specified by this parameter; otherwise, an exception is thrown.	GetObjectRequest.setIfUnmodifiedSince
If-Match	Returns the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.	GetObjectRequest.setIfMatchTag
If-None-Match	Returns the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.	GetObjectRequest.setIfNoneMatchTag

NOTE

- The ETag of an object is the MD5 check value of the object.
- If a request includes **If-Unmodified-Since** or **If-Match** and the specified condition is not met, **412 Precondition Failed** will be returned.
- If a request includes **If-Modified-Since** or **If-None-Match**, and the specified condition is not met, **304 Not Modified** will be returned.

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectname");
request.setRangeStart(0l);
request.setRangeEnd(1000l);

request.setIfModifiedSince(new SimpleDateFormat("yyyy-MM-dd").parse("2016-01-01"));
ObsObject obsObject = obsClient.getObject(request);

obsObject.getObjectContent().close();
```

7.6 Rewriting Response Headers

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

When downloading an object, you can rewrite some HTTP/HTTPS response headers. The following table lists rewritable response headers.

Parameter	Description	Method in OBS Android SDK
contentType	Rewrites **Content-Type** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setContentType
contentLanguage	Rewrites **Content-Language** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setContentLanguage
expires	Rewrites **Expires** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setExpires
cacheControl	Rewrites **Cache-Control** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setCacheControl
contentDisposition	Rewrites **Content-Disposition** in HTTP/HTTPS responses.	ObjectRepleaceMetadata.setContentDisposition

Parameter	Description	Method in OBS Android SDK
contentEncoding	Rewrites **Content-Encoding** in HTTP/HTTPS responses.	ObjectRepleaceMetada-ta.setContentEncoding

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectname");
ObjectRepleaceMetadata replaceMetadata = new ObjectRepleaceMetadata();
replaceMetadata.setContentType("image/jpeg");
request.setReplaceMetadata(replaceMetadata);

ObsObject obsObject = obsClient.getObject(request);
Log.i("GetObject", object.getMetadata().getContentType());

obsObject.getObjectContent().close();
```

7.7 Obtaining Customized Metadata

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

After an object is successfully downloaded, its customized data is returned. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Upload the object and customize the metadata.
PutObjectRequest request = new PutObjectRequest("bucketname", "objectname");
ObjectMetadata metadata = new ObjectMetadata();
metadata.addUserMetadata("property", "property-value");
request.setMetadata(metadata);
obsClient.putObject(request);


// Download the object and obtain the customized metadata.
ObsObject obsObject = obsClient.getObject("bucketname", "objectname");
Log.i("GetObject", obsObject.getMetadata().getUserMetadata("property"));

obsObject.getObjectContent().close();
```


7.8 Downloading an Archive Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

If you want to download an Archive object, you need to restore the object first. Two restore options are supported, as described in the following table.

Option	Description	Value in OBS Android SDK
Expedited	Data can be restored within 1 to 5 minutes.	RestoreTierEnum.EXPEDITED
Standard	Data can be restored within 3 to 5 hours. This is the default option.	RestoreTierEnum.STANDARD

You can call **ObsClient.restoreObject** to restore an Archive object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

RestoreObjectRequest request = new RestoreObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("objectname");
request.setDays(1);
request.setRestoreTier(RestoreTierEnum.EXPEDITED);
obsClient.restoreObject(request);

// Wait until the object is restored.
Thread.sleep(60 * 6 * 1000);

// Download an object.
ObsObject obsObject = obsClient.getObject("bucketname", "objectname");

obsObject.getObjectContent().close();
```

NOTE

- The object specified in **ObsClient.restoreObject** must be in the OBS Archive storage class. Otherwise, an exception will be thrown when you call this API.
- **RestoreObjectRequest.setDays** specifies the retention period of restored object, ranging from 1 to 30.
- **RestoreObjectRequest.setTier** specifies the restore option, which indicates the time spent on restoring an object.

7.9 Performing a Resumable Download

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Downloading large files often fails due to poor network conditions or program breakdowns. It is a waste of resources to restart the download process upon a download failure, and the restarted download process may still suffer from the unstable network. To resolve such issues, you can use the API for resumable download, whose working principle is to divide the to-be-downloaded file into multiple parts and download them separately. The download result of each part is recorded in a checkpoint file in real time. Only when all parts are successfully downloaded, the result indicating a successful download will be returned. Otherwise, an exception is thrown to remind you of calling the API again for re-downloading. Based on the download status of each part recorded in the checkpoint file, the re-downloading will download the parts failed to be downloaded previously, instead of downloading all parts. By virtue of this, resources are saved and efficiency is improved.

You can call **ObsClient.downloadFile** to perform a resumable download. The following table describes the parameters involved in this API.

Parameter	Description	Method in OBS Android SDK
bucketName	(Mandatory) Bucket name	DownloadFileRequest.setBucketName
objectKey	(Mandatory) Object name	DownloadFileRequest.setObjectKey
downloadFile	Local path to which the object is downloaded. If this parameter is null, the downloaded object is saved in the directory where the program is executed.	DownloadFileRequest.setDownloadFile
partSize	Part size, in bytes. The value ranges from 100 KB to 5 GB and defaults to 5 MB.	DownloadFileRequest.setPartSize
taskNum	Maximum number of threads that can be concurrently executed for downloading. The default value is **1**.	DownloadFileRequest.setTaskNum

Parameter	Description	Method in OBS Android SDK
enableCheckpoint	Whether to enable the resumable download mode. The default value is **false**, which indicates that this mode is disabled.	DownloadFileRequest.setEnableCheckpoint
checkpointFile	File used to record the download progress. This parameter is effective only in the resumable download mode. If this parameter is null, the file will be in the same local directory as the downloaded object.	DownloadFileRequest.setCheckpointFile
versionId	Object version	DownloadFileRequest.setVersionId
ifModifiedSince	Returns the object if it is modified after the time specified by this parameter; otherwise, an exception is thrown.	DownloadFileRequest.setIfModifiedSince
ifUnmodifiedSince	Returns the object if it remains unchanged since the time specified by this parameter; otherwise, an exception is thrown.	DownloadFileRequest.setIfUnmodifiedSince
ifMatchTag	Returns the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.	DownloadFileRequest.setIfMatchTag
ifNoneMatchTag	Returns the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.	DownloadFileRequest.setIfNoneMatchTag
progressListener	Configure the data transmission listener to obtain download progresses.	DownloadFileRequest.setProgressListener

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
```



```
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
DownloadFileRequest request = new DownloadFileRequest("bucketname", "objectname");
// Set the local path to which the object is downloaded.
request.setDownloadFile("localfile");
// Set the maximum number of threads that can be concurrently executed for downloading.
request.setTaskNum(5);
// Set the part size to 10 MB.
request.setPartSize(10 * 1024 * 1024);
// Enable resumable download.
request.setEnableCheckpoint(true);
try{
    // Perform a resumable download.
    DownloadFileResult result = obsClient.downloadFile(request);
}catch (ObsException e) {
// When an exception occurs, you can call the API for resumable download again to perform re-downloading.
}
```

NOTE

- The API for resumable download, which is implemented based on **partial download**, is an encapsulated and enhanced version of partial download.
- This API saves resources and improves efficiency upon the re-download, and speeds up the download process by concurrently downloading parts. Because this API is transparent to users, users are free from concerns about internal service details, such as the creation and deletion of checkpoint files, division of objects, and concurrent download of parts.
- The default value of the **enableCheckpoint** parameter is **false**, which indicates that the resumable download mode is disabled. In such cases, this API degrades to the simple encapsulation of partial download, and no checkpoint file will be generated.
- **checkpointFile** is effective only when **enableCheckpoint** is **true**.

7.10 Processing an Image

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

OBS can be used to process images in a stable, secure, efficient, easy to use, and cost-efficient manner. If the object to be downloaded is an image, you can pass the image processing parameters to process the image, including cutting and resizing it as well as putting a watermark and converting the format.

For more information, see the **Image Processing Feature Guide**.

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);
GetObjectRequest request = new GetObjectRequest("bucketname", "objectname.jpg");
// Resize and then rotate the image.
```



```
request.setImageProcess("image/resize,m_fixed,w_100,h_100/rotate,90");
ObsObject obsObject = obsClient.getObject(request);

obsObject.getObjectContent().close();
```

NOTE

- Use **GetObjectRequest.setImageProcess** to specify the image processing parameters.
- Image processing parameters can be processed in cascading mode. This indicates that multiple commands can be performed on an image in sequence.

8 Object Management

8.1 Obtaining Object Properties

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getObjectMetadata** to obtain properties of an object, including the length, MIME type, customized metadata. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectMetadata metadata = obsClient.getObjectMetadata("bucketname", "objectname");
Log.i("GetObjectMetadata","\t" + metadata.getContentType());
Log.i("GetObjectMetadata","\t" + metadata.getContentLength());
Log.i("GetObjectMetadata","\t" + metadata.getUserMetadata("property"));
```

8.2 Managing Object ACLs

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Object ACLs, similar to bucket ACLs, support pre-defined access control policies and direct configuration. For details, see **Managing Bucket ACLs**.

An object **ACL** can be configured in three modes:

1. Specify a pre-defined access control policy during object upload.
2. Call **ObsClient.setObjectAcl** to specify a pre-defined access control policy.
3. Call **ObsClient.setObjectAcl** to set the ACL directly.

Specifying a Pre-defined Access Control Policy During Object Upload

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

PutObjectRequest request = new PutObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("objectname");
request.setFile(new File("localfile")); // localfile indicates the path of the local file to be uploaded. You need to specify the file name.
// Set the object ACL to public-read.
request.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ);
obsClient.putObject(request);
```

Setting a Pre-defined Access Control Policy for an Object

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set the object ACL to private.
obsClient.setObjectAcl("bucketname", "objectname", AccessControlList.REST_CANNED_PRIVATE);
```

Directly Setting an Object ACL

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
owner.setId("ownerid");
acl.setOwner(owner);
// Grant the FULL_CONTROL permission to a specified user.
acl.grantPermission(new CanonicalGrantee("userid"), Permission.PERMISSION_FULL_CONTROL);
// Grant the READ permission to all users.
acl.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
obsClient.setObjectAcl("bucketname", "objectname", acl);
```

NOTE

The owner or grantee ID needed in the ACL indicates the account ID, which can be viewed on the **My Credential** page of OBS Console.

Obtaining an Object ACL

You can call **ObsClient.getObjectAcl** to obtain an object ACL. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = obsClient.getObjectAcl("bucketname", "objectname");
Log.i("GetObjectAcl", acl);
```

8.3 Listing Objects

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.listObjects** to list objects in a bucket.

The following table describes the parameters involved in this API.

Parameter	Description	Method in OBS Android SDK
bucketName	Bucket name	ListObjectsRequest.setBucketName
prefix	Name prefix that the objects to be listed must contain	ListObjectsRequest.setPrefix
marker	Object name to start with when listing objects in a bucket. All objects are listed in the lexicographical order.	ListObjectsRequest.setMarker
maxKeys	Maximum number of objects listed in the response. The value ranges from 1 to 1000. If the value is not in this range, 1000 objects are listed by default.	ListObjectsRequest.setMaxKeys

Parameter	Description	Method in OBS Android SDK
delimiter	Character used to group object names. If the object name contains the **delimiter** parameter, the character string from the first character to the first delimiter in the object name is grouped under a single result element, **commonPrefix**. (If a prefix is specified in the request, the prefix must be removed from the object name.)	ListObjectsRequest.setDelimiter

Listing Objects in Simple Mode

The following sample code shows how to list objects in simple mode. A maximum of 1000 objects can be returned.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ObjectListing result = obsClient.listObjects("bucketname");
for(ObsObject obsObject : result.getObjects()){
    Log.i("ListObjects", "\t" + obsObject.getObjectKey());
    Log.i("ListObjects","\t" + obsObject.getOwner());
}
```

NOTE

- A maximum of 1000 objects can be listed each time. If a bucket contains more than 1000 objects and **ObjectListing.isTruncated** is **true** in the returned result, not all objects are listed. In such cases, you can use **ObjectListing.getNextMarker** to obtain the start position for next listing.
- If you want to obtain all objects in a specified bucket, you can use the paging mode for listing objects.

Listing Objects by Specifying the Number

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// Specify the number of objects to be listed to 100.
request.setMaxKeys(100);
ObjectListing result = obsClient.listObjects(request);
for(ObsObject obsObject : result.getObjects()){
    Log.i("ListObjects","\t" + obsObject.getObjectKey());
```



```
    Log.i("ListObjects","\t" + obsObject.getOwner());
}
```

Listing Objects by Specifying a Prefix

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// Set the number to 100 and the prefix to prefix.
request.setMaxKeys(100);
request.setPrefix("prefix");
ObjectListing result = obsClient.listObjects(request);
for(ObsObject obsObject : result.getObjects()){
    Log.i("ListObjects","\t" + obsObject.getObjectKey());
    Log.i("ListObjects","\t" + obsObject.getOwner());
}
```

Listing Objects by Specifying the Start Position

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// List 100 objects following test in lexicographic order.
request.setMaxKeys(100);
request.setMarker("test");
ObjectListing result = obsClient.listObjects(request);
for(ObsObject obsObject : result.getObjects()){
    Log.i("ListObjects","\t" + obsObject.getObjectKey());
    Log.i("ListObjects","\t" + obsObject.getOwner());
}
```

Listing All Objects in Paging Mode

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// Set the number of objects displayed per page to 100.
request.setMaxKeys(100);

ObjectListing result;
do{
    result = obsClient.listObjects(request);
    for(ObsObject obsObject : result.getObjects()){
        Log.i("ListObjects","\t" + obsObject.getObjectKey());
        Log.i("ListObjects","\t" + obsObject.getOwner());
```



```
    }

    request.setMarker(result.getNextMarker());
}while(result.isTruncated());
```

Listing All Objects in a Folder

There is no folder concept in OBS. All elements in buckets are objects. Folders are actually objects whose sizes are 0 and whose names end with a slash (/). When you set a folder name as the prefix, objects in this folder will be listed. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
// Set the prefix of objects in the folder to dir/.
request.setPrefix("dir/");
request.setMaxKeys(1000);

ObjectListing result;

do{
    result = obsClient.listObjects(request);
    for (ObsObject obsObject : result.getObjects())
    {
        Log.i("ListObjects","\t" + obsObject.getObjectKey());
        Log.i("ListObjects","\t" + obsObject.getOwner());
    }
    request.setMarker(result.getNextMarker());
}while(result.isTruncated());
```

Listing All Objects According to Folders in a Bucket

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListObjectsRequest request = new ListObjectsRequest("bucketname");
request.setMaxKeys(1000);
// Set folder isolators to slashes.
request.setDelimiter("/");
ObjectListing result = obsClient.listObjects(request);
Log.i("ListObjects", "Objects in the root directory:");
for(ObsObject obsObject : result.getObjects()){
    Log.i("ListObjects","\t" + obsObject.getObjectKey());
    Log.i("ListObjects","\t" + obsObject.getOwner());
}
listObjectsByPrefix(obsClient, request, result);
```

The following is the sample code of the **listObjectsByPrefix** function, which is used to recursively list objects in sub-folders.

```
static void listObjectsByPrefix(ObsClient obsClient, ListObjectsRequest request, ObjectListing result) throws
ObsException
{
    for(String prefix : result.getCommonPrefixes()){
        Log.i("ListObjects", "Objects in folder [" + prefix + "]:");
```



```
        request.setPrefix(prefix);
        result  = obsClient.listObjects(request);
        for(ObsObject obsObject : result.getObjects()){
            Log.i("ListObjects","\t" + obsObject.getObjectKey());
            Log.i("ListObjects","\t" + obsObject.getOwner());
        }
        listObjectsByPrefix(obsClient, request, result);
    }
}
```

NOTE

- The sample code does not apply to scenarios where the number of objects in a folder exceeds 1000.
- Because objects and sub-folders in a folder are to be listed and all the objects end with a slash (/), **delimiter** is always a slash (/).
- In the returned result of each recursion, **ObjectListing.getObjects** includes the objects in the folder and **ObjectListing.getCommonPrefixes** includes the sub-folders in the folder.

8.4 Deleting Objects

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

NOTE

Exercise caution when performing this operation. If the versioning function is disabled for the bucket where the object is located, the object cannot be restored after being deleted.

Deleting a Single Object

You can call **ObsClient.deleteObject** to delete a single object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
obsClient.deleteObject("bucketname", "objectname");
```

Deleting Objects in a Batch

You can call **ObsClient.deleteObjects** to delete objects in a batch.

A maximum of 1000 objects can be deleted each time. Two response modes are supported: **verbose** (detailed) and **quiet** (brief).

- In verbose mode (default mode), the returned response includes the deletion result of each requested object.

- In quiet mode, the returned response includes only results of objects failed to be deleted.

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsRequest request = new ListVersionsRequest("bucketname");
// Delete 100 objects at a time.
request.setMaxKeys(100);
ListVersionsResult result;
do {
    result = obsClient.listVersions(request);

    DeleteObjectsRequest deleteRequest = new DeleteObjectsRequest("bucketname");

    for(VersionOrDeleteMarker v  : result.getVersions()) {
        deleteRequest.addKeyAndVersion(v.getKey(), v.getVersionId());
    }

    DeleteObjectsResult deleteResult = obsClient.deleteObjects(deleteRequest);
    // Obtain the successfully deleted objects.
    Log.i("DeletesObjects",deleteResult.getDeletedObjectResults());
    // Obtain the list of objects failed to be deleted.
    Log.i("DeletesObjects", deleteResult.getErrorResults());

    request.setKeyMarker(deleteResult.getNextKeyMarker());
    request.setVersionIdMarker(deleteResult.getNextVersionIdMarker());
}while(result.isTruncated());
```

8.5 Copying an Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

The object copy operation can create a copy for an existing object in OBS.

You can call **ObsClient.copyObject** to copy an object. When copying an object, you can rewrite properties and ACL for it, as well as set restriction conditions.

NOTE

- If the source object is an Archive object, you must restore it before copying it.

Copying an Object in Simple Mode

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
```



```
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectResult result = obsClient.copyObject("sourcebucketname", "sourceobjectname",
"destbucketname", "destobjectname");
Log.i("CopyObject","\t" + result.getEtag());
```

Rewriting Object Properties

The following sample code shows how to rewrite object properties.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest("sourcebucketname", "sourceobjectname",
"destbucketname", "destobjectname");
// Rewrite object properties.
request.setReplaceMetadata(true);
ObjectMetadata newObjectMetadata = new ObjectMetadata();
newObjectMetadata.setContentType("image/jpeg");
newObjectMetadata.addUserMetadata("property", "property-value");
newObjectMetadata.setObjectStorageClass(StorageClassEnum.WARM);
request.setNewObjectMetadata(newObjectMetadata);
CopyObjectResult result = obsClient.copyObject(request);
Log.i("CopyObject","\t" + result.getEtag());
```

NOTE

CopyObjectRequest.setReplaceMetadata and **CopyObjectRequest.setNewObjectMetadata** must be used together.

Copying an Object by Specifying Conditions

When copying an object, you can specify one or more restriction conditions. If the conditions are met, the object will be copied. Otherwise, an exception will be thrown and the copy will fail.

You can set the following conditions:

Parameter	Description	Method in OBS Android SDK
Copy-Source-If-Modified-Since	Copies the source object if it is changed after the time specified by this parameter; otherwise, an exception is thrown.	CopyObjectRequest.setIfModifiedSince
Copy-Source-If-Unmodified-Since	Copies the source object if it is changed before the time specified by this parameter; otherwise, an exception is thrown.	CopyObjectRequest.setIfUnmodifiedSince

Parameter	Description	Method in OBS Android SDK
Copy-Source-If-Match	Copies the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.	CopyObjectRequest.setIfMatchTag
Copy-Source-If-None-Match	Copies the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.	CopyObjectRequest.setIfNoneMatchTag

NOTE

- The ETag of the source object is the MD5 check value of the source object.
- If **Copy-Source-If-Unmodified-Since**, **Copy-Source-If-Match**, **Copy-Source-If-Modified-Since**, or **Copy-Source-If-None-Match** is included and its specified condition is not met, an exception, whose HTTP status code is **412 Precondition Failed**, will be thrown.
- **Copy-Source-If-Modified-Since** and **Copy-Source-If-None-Match** can be used together, and so do **Copy-Source-If-Unmodified-Since** and **Copy-Source-If-Match**.

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest("sourcebucketname", "sourceobjectname",
"destbucketname", "destobjectname");

request.setIfModifiedSince(new SimpleDateFormat("yyyy-MM-dd").parse("2016-01-01"));
request.setIfNoneMatchTag("none-match-etag");

CopyObjectResult result = obsClient.copyObject(request);
Log.i("CopyObject","\t" + result.getEtag());
```

Rewriting an Object ACL

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest("sourcebucketname", "sourceobjectname",
"destbucketname", "destobjectname");

// Modify the Object ACL to public-read.
```



```
request.setAcl(AccessControlList.REST_CANNED_PUBLIC_READ);
CopyObjectResult result = obsClient.copyObject(request);
Log.i("CopyObject","\t" + result.getEtag());
```

9 Temporarily Authorized Access

9.1 Using a Temporary URL for Authorized Access

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

ObsClient allows you to create a URL whose **Query** parameters are carried with authentication information by specifying the AK and SK, HTTP method, and request parameters. You can provide other users with this URL for temporary access. When generating a URL, you need to specify the validity period of the URL to restrict the access duration of visitors.

If you want to grant other users the permission to perform other operations on buckets or objects (for example, upload or download objects), generate a URL with the corresponding request (for example, to upload an object using the URL that generates the PUT request) and provide the URL for other users.

The following table lists operations can be performed through a signed URL.

Operation	HTTP Request Method (Value in OBS Android SDK)	Special Operator (Value in OBS Android SDK)	Bucket Name Required	Object Name Required
PUT Bucket	HttpMethodEnum.PUT	N/A	Yes	No
GET Buckets	HttpMethodEnum.GET	N/A	No	No
DELETE Bucket	HttpMethodEnum.DELETE	N/A	Yes	No

Operation	HTTP Request Method (Value in OBS Android SDK)	Special Operator (Value in OBS Android SDK)	Bucket Name Required	Object Name Required
GET Objects	HttpMethodEnum.GET	N/A	Yes	No
GET Object versions	HttpMethodEnum.GET	SpecialParamEnum.VERSIONS	Yes	No
List Multipart uploads	HttpMethodEnum.GET	SpecialParamEnum.UPLOADS	Yes	No
Obtain Bucket Metadata	HttpMethodEnum.HEAD	N/A	Yes	No
GET Bucket location	HttpMethodEnum.GET	SpecialParamEnum.LOCATION	Yes	No
GET Bucket storageinfo	HttpMethodEnum.GET	SpecialParamEnum.STORAGEINFO	Yes	No
PUT Bucket quota	HttpMethodEnum.PUT	SpecialParamEnum.QUOTA	Yes	No
GET Bucket quota	HttpMethodEnum.GET	SpecialParamEnum.QUOTA	Yes	No
Set Bucket storagePolicy	HttpMethodEnum.PUT	SpecialParamEnum.STORAGEPOLICY	Yes	No
GET Bucket storagePolicy	HttpMethodEnum.GET	SpecialParamEnum.STORAGEPOLICY	Yes	No
PUT Bucket acl	HttpMethodEnum.PUT	SpecialParamEnum.ACL	Yes	No
GET Bucket acl	HttpMethodEnum.GET	SpecialParamEnum.ACL	Yes	No
PUT Bucket logging	HttpMethodEnum.PUT	SpecialParamEnum.LOGGING	Yes	No
GET Bucket logging	HttpMethodEnum.GET	SpecialParamEnum.LOGGING	Yes	No
PUT Bucket policy	HttpMethodEnum.PUT	SpecialParamEnum.POLICY	Yes	No
GET Bucket policy	HttpMethodEnum.GET	SpecialParamEnum.POLICY	Yes	No

Operation	HTTP Request Method (Value in OBS Android SDK)	Special Operator (Value in OBS Android SDK)	Bucket Name Required	Object Name Required
DELETE Bucket policy	HttpMethodEnum.DELETE	SpecialParamEnum.POLICY	Yes	No
PUT Bucket lifecycle	HttpMethodEnum.PUT	SpecialParamEnum.LIFECYCLE	Yes	No
GET Bucket lifecycle	HttpMethodEnum.GET	SpecialParamEnum.LIFECYCLE	Yes	No
DELETE Bucket lifecycle	HttpMethodEnum.DELETE	SpecialParamEnum.LIFECYCLE	Yes	No
PUT Bucket website	HttpMethodEnum.PUT	SpecialParamEnum.WEBSITE	Yes	No
GET Bucket website	HttpMethodEnum.GET	SpecialParamEnum.WEBSITE	Yes	No
DELETE Bucket website	HttpMethodEnum.DELETE	SpecialParamEnum.WEBSITE	Yes	No
PUT Bucket versioning	HttpMethodEnum.PUT	SpecialParamEnum.VERSIONING	Yes	No
GET Bucket versioning	HttpMethodEnum.GET	SpecialParamEnum.VERSIONING	Yes	No
PUT Bucket cors	HttpMethodEnum.PUT	SpecialParamEnum.CORS	Yes	No
GET Bucket cors	HttpMethodEnum.GET	SpecialParamEnum.CORS	Yes	No
DELETE Bucket cors	HttpMethodEnum.DELETE	SpecialParamEnum.CORS	Yes	No
PUT Bucket notification	HttpMethodEnum.PUT	SpecialParamEnum.NOTIFICATION	Yes	No
GET Bucket notification	HttpMethodEnum.GET	SpecialParamEnum.NOTIFICATION	Yes	No
PUT Bucket tagging	HttpMethodEnum.PUT	SpecialParamEnum.TAGGING	Yes	No
GET Bucket tagging	HttpMethodEnum.GET	SpecialParamEnum.TAGGING	Yes	No

Operation	HTTP Request Method (Value in OBS Android SDK)	Special Operator (Value in OBS Android SDK)	Bucket Name Required	Object Name Required
DELETE Bucket tagging	HttpMethodEnum.DELETE	SpecialParamEnum.TAGGING	Yes	No
PUT Object	HttpMethodEnum.PUT	N/A	Yes	Yes
Append Object	HttpMethodEnum.POST	SpecialParamEnum.APPEND	Yes	Yes
GET Object	HttpMethodEnum.GET	N/A	Yes	Yes
PUT Object - Copy	HttpMethodEnum.PUT	N/A	Yes	Yes
DELETE Object	HttpMethodEnum.DELETE	N/A	Yes	Yes
DELETE Objects	HttpMethodEnum.POST	SpecialParamEnum.DELETE	Yes	Yes
Obtain Object Metadata	HttpMethodEnum.HEAD	N/A	Yes	Yes
PUT Object acl	HttpMethodEnum.PUT	SpecialParamEnum.ACL	Yes	Yes
GET Object acl	HttpMethodEnum.GET	SpecialParamEnum.ACL	Yes	Yes
Initiate Multipart Upload	HttpMethodEnum.POST	SpecialParamEnum.UPLOADS	Yes	Yes
PUT Part	HttpMethodEnum.PUT	N/A	Yes	Yes
PUT Part - Copy	HttpMethodEnum.PUT	N/A	Yes	Yes
List Parts	HttpMethodEnum.GET	N/A	Yes	Yes
Complete Multipart Upload	HttpMethodEnum.POST	N/A	Yes	Yes
DELETE Multipart upload	HttpMethodEnum.DELETE	N/A	Yes	Yes

Operation	HTTP Request Method (Value in OBS Android SDK)	Special Operator (Value in OBS Android SDK)	Bucket Name Required	Object Name Required
POST Object restore	HttpMethodEnum.POST	SpecialParamEnum.RESTORE	Yes	Yes

To access OBS using a temporary URL generated by the OBS Android SDK, perform the following steps:

Step 1 Call **ObsClient.createTemporarySignature** to generate a signed URL.

Step 2 Use any HTTP library to make an HTTP/HTTPS request to OBS.

----End

The following content provides examples of accessing OBS using a temporary URL, including bucket creation, as well as object upload, download, listing, and deletion.

Creating a Bucket

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;

TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.PUT,
expireSeconds);
request.setBucketName("bucketname");
TemporarySignatureResponse response = obsClient.createTemporarySignature(request);
Log.i("CreateTemporarySignature", "Creating bucket using temporary signature url:");
Log.i("CreateTemporarySignature", "\t" + response.getSignedUrl());

Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
      builder.header(entry.getKey(), entry.getValue());
}
// Make a PUT request to create a bucket.
String location = "your bucket location";
Request httpRequest = builder.url(response.getSignedUrl()).put(RequestBody.create(null,
"<CreateBucketConfiguration><LocationConstraint>" + location + "</LocationConstraint></
CreateBucketConfiguration>".getBytes())).build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
            .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
Log.i("CreateTemporarySignature", "\tStatus:" + res.code());
if (res.body() != null) {
      Log.i("CreateTemporarySignature", "\tContent:" + res.body().string() + "\n");
}
res.close();
```


Uploading an Object

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;

Map<String, String> headers = new HashMap<String, String>();
String contentType = "text/plain";
headers.put("Content-Type", contentType);

TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.PUT,
expireSeconds);
request.setBucketName("bucketname");
request.setObjectKey("objectname");
request.setHeaders(headers);

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

Log.i("CreateTemporarySignature", "Creating object using temporary signature url:");
Log.i("CreateTemporarySignature", "\t" + response.getSignedUrl());
Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
    builder.header(entry.getKey(), entry.getValue());
}

//Make a PUT request to upload an object.
Request httpRequest =
builder.url(response.getSignedUrl()).put(RequestBody.create(MediaType.parse(contentType), "Hello
OBS".getBytes("UTF-8"))).build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
        .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
Log.i("CreateTemporarySignature", "\tStatus:" + res.code());
if (res.body() != null) {
    Log.i("CreateTemporarySignature", "\tContent:" + res.body().string() + "\n");
}
res.close();
```

Downloading an Object

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;


TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.GET,
expireSeconds);
request.setBucketName("bucketname");
request.setObjectKey("objectname");

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

Log.i("CreateTemporarySignature", "Getting object using temporary signature url:");
Log.i("CreateTemporarySignature", "\t" + response.getSignedUrl());
Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
```



```
    builder.header(entry.getKey(), entry.getValue());
}

//Make a GET request to download an object.
Request httpRequest = builder.url(response.getSignedUrl()).get().build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
        .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
Log.i("CreateTemporarySignature", "\tStatus:" + res.code());
if (res.body() != null) {
    Log.i("CreateTemporarySignature", "\tContent:" + res.body().string() + "\n");
}
res.close();
```

Listing Objects

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;


TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.GET,
expireSeconds);
request.setBucketName("bucketname");

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

Log.i("CreateTemporarySignature", "Getting object list using temporary signature url:");
Log.i("CreateTemporarySignature", "\t" + response.getSignedUrl());
Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
    builder.header(entry.getKey(), entry.getValue());
}

//Make a GET request to obtain the object list.
Request httpRequest = builder.url(response.getSignedUrl()).get().build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
        .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
Log.i("CreateTemporarySignature", "\tStatus:" + res.code());
if (res.body() != null) {
    Log.i("CreateTemporarySignature", "\tContent:" + res.body().string() + "\n");
}
res.close();
```

Deleting an Object

```
String endPoint = "http://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
// Specify the validity period of the URL to 3600 seconds.
long expireSeconds = 3600L;


TemporarySignatureRequest request = new TemporarySignatureRequest(HttpMethodEnum.DELETE,
```



```
expireSeconds);
request.setBucketName("bucketname");
request.setObjectKey("objectname");

TemporarySignatureResponse response = obsClient.createTemporarySignature(request);

Log.i("CreateTemporarySignature", "Deleting object using temporary signature url:");
Log.i("CreateTemporarySignature", "\t" + response.getSignedUrl());
Request.Builder builder = new Request.Builder();
for (Map.Entry<String, String> entry : response.getActualSignedRequestHeaders().entrySet()) {
    builder.header(entry.getKey(), entry.getValue());
}

//Make a DELETE request to delete an object.
Request httpRequest = builder.url(response.getSignedUrl()).delete().build();
OkHttpClient httpClient = new
OkHttpClient.Builder().followRedirects(false).retryOnConnectionFailure(false)
        .cache(null).build();

Call c = httpClient.newCall(httpRequest);
Response res = c.execute();
Log.i("CreateTemporarySignature", "\tStatus:" + res.code());
if (res.body() != null) {
    Log.i("CreateTemporarySignature", "\tContent:" + res.body().string() + "\n");
}
res.close();
```

NOTE

HttpMethodEnum is an enumeration function defined in OBS Android SDK, whose value indicates the request method types.

10 Versioning Management

10.1 Versioning Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

OBS can store multiple versions of an object. You can quickly search for and restore different versions as well as restore data in the event of misoperations or application faults.

For details, see **Versioning**.

10.2 Setting Versioning Status for a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.setBucketVersioning** to set the versioning status for a bucket. OBS supports two versioning statuses.

Versioning Status	Description	Value in OBS Android SDK
Enabled	1. OBS creates a unique version ID for each uploaded object. Namesake objects are not overwritten and are distinguished by their own version IDs. 2. Objects can be downloaded by specifying the version ID. By default, the object of the latest version is downloaded if no version ID is specified. 3. Objects can be deleted by specifying the version ID. If an object is deleted with no version ID specified, the object will generate a delete marker with a unique version ID but is not physically deleted. 4. The latest objects in a bucket are returned by default after **ObsClient.listObjects** is called. You can call **ObsClient.listVersions** to list a bucket's objects with all version IDs. 5. Except for delete markers, storage space occupied by objects with all version IDs is billed.	VersioningStatusEnum.ENABLED
Suspended	1. Noncurrent object versions are not affected. 2. OBS creates version ID **null** to an uploaded object and the object will be overwritten after a namesake one is uploaded. 3. Objects can be downloaded by specifying the version ID. By default, the object of the latest version is downloaded if no version ID is specified. 4. Objects can be deleted by specifying version IDs. If an object is deleted with no version ID specified, the object is only attached with a delete marker whose version ID is **null**. Objects with version ID **null** are physically deleted. 5. Except for delete markers, storage space occupied by objects with all version IDs is billed.	VersioningStatusEnum.SUSPENDED

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Enable versioning.
obsClient.setBucketVersioning("bucketname", new
BucketVersioningConfiguration(VersioningStatusEnum.ENABLED));

// Suspend versioning.
obsClient.setBucketVersioning("bucketname", new
BucketVersioningConfiguration(VersioningStatusEnum.SUSPENDED));
```

10.3 Viewing Versioning Status of a Bucket

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getBucketVersioning** to view the versioning status of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketVersioningConfiguration status = obsClient.getBucketVersioning("bucketname");
Log.i("GetBucketVersioning", "\t" + status.getVersioningStatus());
```

10.4 Obtaining a Versioning Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getObject** to pass the version ID (**versionId**) to obtain a versioning object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
```



```
// Set versionId to obtain a versioning object.
ObsObject obsObject = obsClient.getObject("bucketname", "objectname", "versionid");
obsObject.getObjectContent().close();
```

NOTE

If version ID is null, the object of the latest version will be downloaded, by default.

10.5 Copying a Versioning Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.copyObject** to pass the version ID (**versionId**) to copy a versioning object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

CopyObjectRequest request = new CopyObjectRequest();
request.setSourceBucketName("sourebucketname");
request.setSourceObjectKey("sourceobjectname");
// Set the version ID of the object to be copied.
request.setVersionId("versionid");
request.setDestinationBucketName("destbucketname");
request.setDestinationObjectKey("destobjectname");
obsClient.copyObject(request);
```

10.6 Restoring a Versioning Archive Object

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.restoreObject** to pass the version ID (**versionId**) to restore a versioning Archive object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

RestoreObjectRequest request = new RestoreObjectRequest("bucketname", "objectname", 1);
// Restore a versioning object in the Expedited mode.
request.setRestoreTier(RestoreTierEnum.EXPEDITED);
request.setVersionId("versionid");
obsClient.restoreObject(request);
```


10.7 Listing Versioning Objects

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.listVersions** to list versioning objects.

The following table describes the parameters involved in this API.

Parameter	Description
bucketName	Bucket name
prefix	Name prefix that the objects to be listed must contain
keyMarker	Object name to start with when listing versioning objects in a bucket. All versioning objects whose names follow this parameter are listed in the lexicographical order.
maxKeys	Maximum number of versioning objects listed. The value ranges from 1 to 1000. If the value is not in this range, 1000 versioning objects are listed by default.
delimiter	Character used to group object names. If the object name contains the **delimiter** parameter, the character string from the first character to the first delimiter in the object name is grouped under a single result element, **commonPrefix**. (If a prefix is specified in the request, the prefix must be removed from the object name.)
versionIdMarker	All versioning objects are listed in the lexicographical order by object name and version ID. This parameter must be used together with **keyMarker**.

NOTE

- If the value of **versionIdMarker** is not a version ID specified by **keyMarker**, **versionIdMarker** is ineffective.
- The returned result of **ObsClient.listVersions** includes the versioning objects and delete markers.

Listing Versioning Objects in Simple Mode

The following sample code shows how to list versioning objects in simple mode. A maximum of 1000 versioning objects can be returned.

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
```



```
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result = obsClient.listVersions("bucketname");

for(VersionOrDeleteMarker v : result.getVersions()){
    Log.i("ListVersions", "\t" + v.getKey());
    Log.i("ListVersions","\t" + v.getOwner());
    Log.i("ListVersions","\t" + v.isDeleteMarker());
}
```

NOTE

- A maximum of 1000 versioning objects can be listed each time. If a bucket contains more than 1000 objects and **ListVersionsResult.isTruncated** is **true** in the returned result, not all versioning objects are listed. In such cases, you can use **ListVersionsResult.getNextKeyMarker** and **ListVersionsResult.getNextVersionIdMarker** to obtain the start position for next listing.
- If you want to obtain all versioning objects in a specified bucket, you can use the paging mode for listing objects.

Listing Versioning Objects by Specifying the Number

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result = obsClient.listVersions("bucketname", 100);
for(VersionOrDeleteMarker v : result.getVersions()){
    Log.i("ListVersions","\t" + v.getKey());
    Log.i("ListVersions","\t" + v.getOwner());
    Log.i("ListVersions","\t" + v.isDeleteMarker());
}
```

Listing Versioning Objects by Specifying a Prefix

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// List 100 objects whose name prefix is prefix.
ListVersionsRequest request = new ListVersionsRequest ("bucketname", 100);
request.setPrefix("prefix");
ListVersionsResult result = obsClient.listVersions(request);
for(VersionOrDeleteMarker v : result.getVersions()){
    Log.i("ListVersions","\t" + v.getKey());
    Log.i("ListVersions","\t" + v.getOwner());
    Log.i("ListVersions","\t" + v.isDeleteMarker());
}
```

Listing Versioning Objects by Specifying the Start Position

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
```



```
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// List 100 versioning objects whose names following test in lexicographic order.
ListVersionsRequest request = new ListVersionsRequest ("bucketname", 100);
request.setKeyMarker("test");
ListVersionsResult result = obsClient.listVersions(request);

for(VersionOrDeleteMarker v : result.getVersions()){
    Log.i("ListVersions","\t" + v.getKey());
    Log.i("ListVersions","\t" + v.getOwner());
    Log.i("ListVersions","\t" + v.isDeleteMarker());
}
```

Listing All Versioning Objects in Paging Mode

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result;
ListVersionsRequest request = new ListVersionsRequest ("bucketname", 100);
do{
    result = obsClient.listVersions(request);
    for(VersionOrDeleteMarker v : result.getVersions()){
        Log.i("ListVersions","\t" + v.getKey());
        Log.i("ListVersions","\t" + v.getOwner());
        Log.i("ListVersions","\t" + v.isDeleteMarker());
    }
    request.setKeyMarker(result.getNextKeyMarker());
    request.setVersionIdMarker(result.getNextVersionIdMarker());
}while(result.isTruncated());
```

Listing All Versioning Objects in a Folder

There is no folder concept in OBS. All elements in buckets are objects. Folders are actually objects whose sizes are 0 and whose names end with a slash (/). When you set a folder name as the prefix, objects in this folder will be listed. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

ListVersionsResult result;
ListVersionsRequest request = new ListVersionsRequest ("bucketname", 100);
// Set the prefix of objects in the folder to dir/.
request.setPrefix("dir/");
do{
    result = obsClient.listVersions(request);
    for(VersionOrDeleteMarker v : result.getVersions()){
        Log.i("ListVersions","\t" + v.getKey());
        Log.i("ListVersions","\t" + v.getOwner());
        Log.i("ListVersions","\t" + v.isDeleteMarker());
    }
    request.setKeyMarker(result.getNextKeyMarker());
    request.setVersionIdMarker(result.getNextVersionIdMarker());
}while(result.isTruncated());
```


Listing All Versioning Objects According to Folders in a Bucket

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
ListVersionsRequest request = new ListVersionsRequest ("bucketname", 1000);
request.setDelimiter("/");
ListVersionsResult result = obsClient.listVersions(request);
Log.i("ListVersions", "Objects in the root directory:");
for(VersionOrDeleteMarker v : result.getVersions()){
    Log.i("ListVersions","\t" + v.getKey());
    Log.i("ListVersions","\t" + v.getOwner());
    Log.i("ListVersions","\t" + v.isDeleteMarker());
}

listVersionsByPrefix(obsClient, result);
```

The following is the sample code of the **listVersionsByPrefix** function, which is used to recursively list objects in sub-folders.

```
static void listVersionsByPrefix(ObsClient obsClient, ListVersionsResult result) throws ObsException{
    for(String prefix : result.getCommonPrefixes()){
        Log.i("ListVersions", "Objects in folder [" + prefix + "]:");
        ListVersionsRequest request = new ListVersionsRequest ("bucketname", 1000);
        request.setDelimiter("/");
        request.setPrefix(prefix)
        result = obsClient.listVersions(request);
        for(VersionOrDeleteMarker v : result.getVersions()){
            Log.i("ListVersions","\t" + v.getKey());
            Log.i("ListVersions","\t" + v.getOwner());
            Log.i("ListVersions","\t" + v.isDeleteMarker());
        }
        listVersionsByPrefix(obsClient, result);
    }
}
```

NOTE

- The previous sample code does not include scenarios where the number of objects in a folder exceeds 1000.
- Because objects and sub-folders in a folder are to be listed and all the objects end with a slash (/), **delimiter** is always a slash (/).
- In the returned result of each recursion, **ListVersionsResult.getVersions** includes the versioning objects in the folder and **ListVersionsResult.getCommonPrefixes** includes the sub-folders in the folder.

10.8 Setting or Obtaining a Versioning Object ACL

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Directly Setting a Versioning Object ACL

You can call **ObsClient.setObjectAcl** to pass the version ID (**versionId**) to set the ACL for a versioning object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Set the versioning object ACL to public-read by specifying the pre-defined access control policy.
obsClient.setObjectAcl("bucketname", "objectname", AccessControlList.REST_CANNED_PUBLIC_READ,
"versionid");

AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
owner.setId("ownerid");
acl.setOwner(owner);
// Grant the READ permission to all users.
acl.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
// Directly set the versioning object ACL.
obsClient.setObjectAcl("bucketname", "objectname", acl, "versionid");
```

NOTE

The owner or grantee ID needed in the ACL indicates the account ID, which can be viewed on the **My Credential** page of OBS Console.

Obtaining a Versioning Object ACL

You can call **ObsClient.getObjectAcl** to pass the version ID (**versionId**) to obtain the ACL of a versioning object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

AccessControlList acl = obsClient.getObjectAcl("bucketname", "objectname", "versionid");
System.out.println(acl);
```

10.9 Deleting Versioning Objects

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Deleting a Single Versioning Object

You can call **ObsClient.deleteObject** to pass the version ID (**versionId**) to delete a versioning object. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
```



```
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
obsClient.deleteObject("bucketname", "objectname", "versionid");
```

Deleting Versioning Objects in a Batch

You can call **ObsClient.deleteObjects** to pass the version ID (**versionId**) of each to-be-deleted versioning object to delete them at a time. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

DeleteObjectsRequest request = new DeleteObjectsRequest("bucketname");
request.setQuiet(false);
List<KeyAndVersion> toDelete = new ArrayList<KeyAndVersion>();
toDelete.add(new KeyAndVersion("objectname1", "versionid1"));
toDelete.add(new KeyAndVersion("objectname2", "versionid2"));
toDelete.add(new KeyAndVersion("objectname3", "versionid3"));
request.setKeyAndVersions(toDelete.toArray(new KeyAndVersion[toDelete.size()]));
DeleteObjectsResult result = obsClient.deleteObjects(request);

Log.i("DeleteObjects","\t" + result.getDeletedObjectResults());
Log.i("DeleteObjects","\t" + result.getErrorResults());
```


11 Lifecycle Management

11.1 Lifecycle Management Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

OBS allows you to set lifecycle rules for buckets to automatically transit the storage class of an object and delete expired objects, to effectively use storage features and optimize the storage space. You can set multiple lifecycle rules based on the prefix. A lifecycle rule must contain:

- Rule ID, which uniquely identifies the rule
- Prefix of objects that are under the control of this rule
- Transition policy of an object of the latest version, which can be specified in either mode:
  a. How many days after the object is created
  b. Transition date
- Expiration time of an object of the latest version, which can be specified in either mode:
  a. How many days after the object is created
  b. Expiration date
- Transition policy of a noncurrent object version, which can be specified in the following mode:
  – How many days after the object becomes a noncurrent object version
- Expiration time of a noncurrent object version, which can be specified in the following mode:
  – How many days after the object becomes a noncurrent object version
- Identifier specifying whether the setting is effective

For more information, see **Lifecycle Management**.

NOTE

- An object will be automatically deleted by the OBS server once it expires.
- The time set in the transition policy of an object must be earlier than its expiration time, and the time set in the transition policy of a noncurrent object version must be earlier than its expiration time.
- The configured expiration time and transition policy for a noncurrent object version will take effect only when the versioning is enabled or suspended for a bucket.

11.2 Setting Lifecycle Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.setBucketLifecycle** to set lifecycle rules for a bucket.

Setting an Object Transition Policy

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";

// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

LifecycleConfiguration config = new LifecycleConfiguration();
Rule rule = config.new Rule();
rule.setEnabled(true);
rule.setId("rule1");
rule.setPrefix("prefix");
Transition transition = config.new Transition();
// Specify that objects whose names contain the prefix will be transited 30 days after creation.
transition.setDays(30);
// Specify the storage class of the objects after transition.
transition.setObjectStorageClass(StorageClassEnum.STANDARD_IA);
// Specify a date when the objects whose names contain the prefix will be transited.
// transition.setDate(new SimpleDateFormat("yyyy-MM-dd").parse("2018-10-31"));
rule.getTransitions().add(transition);

NoncurrentVersionTransition noncurrentVersionTransition = config.new NoncurrentVersionTransition();
// Specify that objects whose names contain the prefix will be transited 30 days after changing into noncurrent versions.
noncurrentVersionTransition.setDays(30);
// Specify the storage class of the noncurrent object version after transition.
noncurrentVersionTransition.setObjectStorageClass(StorageClassEnum.COLD);
rule.getNoncurrentVersionTransitions().add(noncurrentVersionTransition);

config.addRule(rule);

obsClient.setBucketLifecycle("bucketname", config);
```


Setting an Object Expiration Time

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

LifecycleConfiguration config = new LifecycleConfiguration();

Rule rule = config.new Rule();
rule.setEnabled(true);
rule.setId("rule1");
rule.setPrefix("prefix");
Expiration expiration = config.new Expiration();
// Specify that objects whose names contain the prefix will expire 60 days after creation.
expiration.setDays(60);
// Specify a date when the objects whose names contain the prefix will expire.
// expiration.setDate(new SimpleDateFormat("yyyy-MM-dd").parse("2018-12-31"));
rule.setExpiration(expiration);

NoncurrentVersionExpiration noncurrentVersionExpiration = config.new NoncurrentVersionExpiration();
// Specify that objects whose names contain the prefix will expire 60 days after changing into noncurrent versions.
noncurrentVersionExpiration.setDays(60);
rule.setNoncurrentVersionExpiration(noncurrentVersionExpiration);
config.addRule(rule);

obsClient.setBucketLifecycle("bucketname", config);
```

11.3 Viewing Lifecycle Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getBucketLifecycle** to view lifecycle rules of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

LifecycleConfiguration config = obsClient.getBucketLifecycle("bucketname");

for (Rule rule : config.getRules()) {
    Log.i("GetBucketLifecycleConfiguration",rule.getId());
    Log.i("GetBucketLifecycleConfiguration",rule.getPrefix());
    for(Transition transition : rule.getTransitions()){
        Log.i("GetBucketLifecycleConfiguration",transition.getDays());
        Log.i("GetBucketLifecycleConfiguration",transition.getStorageClass());
    }
    Log.i("GetBucketLifecycleConfiguration",rule.getExpiration() != null ? rule.getExpiration().getDays() : "");
    for(NoncurrentVersionTransition noncurrentVersionTransition : rule.getNoncurrentVersionTransitions()){
        Log.i("GetBucketLifecycleConfiguration",noncurrentVersionTransition.getDays());
        Log.i("GetBucketLifecycleConfiguration",noncurrentVersionTransition.getStorageClass());
    }
```



```
    Log.i("GetBucketLifecycleConfiguration",rule.getNoncurrentVersionExpiration() != null ?
rule.getNoncurrentVersionExpiration().getDays() : "");
}
```

11.4 Deleting Lifecycle Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.deleteBucketLifecycle** to delete lifecycle rules of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.deleteBucketLifecycle("bucketname");
```


12 CORS

12.1 CORS Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Cross-origin resource sharing (CORS) allows web application programs to access resources in other domains. OBS provides developers with APIs for facilitating cross-origin resource access.

For more information, see **CORS**.

12.2 Setting CORS Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.setBucketCors** to set CORS rules for a bucket. If the bucket is configured with CORS rules, the newly set ones will overwrite the existing ones. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketCors cors = new BucketCors();
```



```
List<BucketCorsRule> rules = new ArrayList<BucketCorsRule>();
BucketCorsRule rule = new BucketCorsRule();

ArrayList<String> allowedOrigin = new ArrayList<String>();
// Specify the origin of the cross-domain request.
allowedOrigin.add( "http://www.a.com");
allowedOrigin.add( "http://www.b.com");
rule.setAllowedOrigin(allowedOrigin);

ArrayList<String> allowedMethod = new ArrayList<String>();
// Specify the request method, which can be GET, PUT, DELETE, POST, or HEAD.
allowedMethod.add("GET");
allowedMethod.add("HEAD");
allowedMethod.add("PUT");
rule.setAllowedMethod(allowedMethod);

ArrayList<String> allowedHeader = new ArrayList<String>();
// Specify whether headers specified in Access-Control-Request-Headers in the OPTIONS request can be used.
allowedHeader.add("x-obs-header");
rule.setAllowedHeader(allowedHeader);

ArrayList<String> exposeHeader = new ArrayList<String>();
// Specify response headers that users can access using application programs.
exposeHeader.add("x-obs-expose-header");
rule.setExposeHeader(exposeHeader);

// Specify the browser's cache time of the returned results of OPTIONS requests for specific resources, in seconds.
rule.setMaxAgeSecond(10);
rules.add(rule);
cors.setRules(rules);

obsClient.setBucketCors("bucketname", cors);
```

NOTE

AllowedOrigins and **AllowedHeaders** respectively can contain up to one wildcard character (*). The wildcard character (*) indicates that all origins or headers are allowed.

12.3 Viewing CORS Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getBucketCors** to view CORS rules of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);


BucketCors cors = obsClient.getBucketCors("bucketname");
for(BucketCorsRule rule : cors.getRules()){
    Log.i("GetBucketCors","\t" + rule.getId());
    Log.i("GetBucketCors","\t" + rule.getMaxAgeSecond());
```



```
    Log.i("GetBucketCors","\t" + rule.getAllowedHeader());
    Log.i("GetBucketCors","\t" + rule.getAllowedOrigin());
    Log.i("GetBucketCors","\t" + rule.getAllowedMethod());
    Log.i("GetBucketCors","\t" + rule.getExposeHeader());
}
```

12.4 Deleting CORS Rules

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.deleteBucketCors** to delete CORS rules of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.deleteBucketCors("bucketname");
```


13 Access Logging

13.1 Logging Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

OBS allows you to configure access logging for buckets. After the configuration, access to buckets will be recorded in the format of logs. These logs will be saved in specific buckets in OBS.

For more information, see **Logging**.

13.2 Enabling Bucket Logging

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.setBucketLogging** to enable bucket logging.

NOTICE

The source bucket and target bucket of logging must be in the same region.

NOTE

If the bucket is in the OBS Infrequent Access or Archive storage class, it cannot be used as the target bucket.

Enabling Bucket Logging

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketLoggingConfiguration config = new BucketLoggingConfiguration();
config.setAgency("your agency");
config.setTargetBucketName("targetbucketname");
config.setLogfilePrefix("targetprefix");

obsClient.setBucketLogging("bucketname", config);
```

Setting ACLs for Objects to Be Logged

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

String targetBucket = "targetbucketname";

// Configure logging.
BucketLoggingConfiguration config = new BucketLoggingConfiguration();
config.setAgency("your agency");
config.setTargetBucketName(targetBucket);
config.setLogfilePrefix("prefix");

// Grant the READ permission on the objects to be logged to all users.
GrantAndPermission grant1 = new GrantAndPermission(GroupGrantee.ALL_USERS,
Permission.PERMISSION_READ);
config.setTargetGrants(new GrantAndPermission[]{grant1});

obsClient.setBucketLogging("bucketname", config, true);
```

13.3 Viewing Bucket Logging

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getBucketLogging** to view the logging settings of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
```



```
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketLoggingConfiguration config = obsClient.getBucketLogging("bucketname");
Log.i("GetBucketLoggingConfiguration","\t" + config.getTargetBucketName());
Log.i("GetBucketLoggingConfiguration","\t" + config.getLogfilePrefix());
```

13.4 Disabling Bucket Logging

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.setBucketLogging** to clear logging settings of a bucket so as to disable logging of the bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Leave the logging settings in blank.
obsClient.setBucketLogging("bucketname", new BucketLoggingConfiguration());
```


14 Static Website Hosting

14.1 Static Website Hosting Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can upload the content files of the static website to your bucket in OBS as objects and configure the **public-read** permission on the files, and then configure the static website hosting mode for your bucket to host your static websites in OBS. After this, when third-party users access your websites, they actually access the objects in your bucket in OBS. When using static website hosting, you can configure request redirection to redirect specific or all requests.

For more information, see **Static Website Hosting**.

14.2 Website File Hosting

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can perform the following to implement website file hosting:

Step 1 Upload a website file to your bucket in OBS as an object and set the MIME type for the object.

Step 2 Set the ACL of the object to **public-read**.

Step 3 Access the object using a browser.

----End

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Upload objects and set the MIME type for the objects.
PutObjectRequest request = new PutObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("test.html");
request.setFile(new File("localfile.html"));
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("text/html");
request.setMetadata(metadata);
obsClient.putObject(request);

// Set the object ACL to public-read.
obsClient.setObjectAcl("bucketname", "test.html", AccessControlList.REST_CANNED_PUBLIC_READ);
```

NOTE

You can use **https://***bucketname.your-endpoint***/test.html** in a browser to access files hosted using the sample code.

14.3 Setting Website Hosting

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.setBucketWebsite** to set website hosting for a bucket.

Configuring the Default Homepage and Error Pages

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

WebsiteConfiguration config = new WebsiteConfiguration();
// Configure the default homepage.
config.setSuffix("index.html");
// Configure the error pages.
config.setKey("error.html");
obsClient.setBucketWebsite("bucketname", config);
```

Configuring the Redirection Rules

Sample code:


```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

WebsiteConfiguration config = new WebsiteConfiguration();
// Configure the default homepage.
config.setSuffix("index.html");
// Configure the error pages.
config.setKey("error.html");

RouteRule rule = new RouteRule();
Redirect r = new Redirect();
r.setHostName("www.example.com");
r.setHttpRedirectCode("305");
r.setRedirectProtocol(ProtocalEnum.HTTP);
r.setReplaceKeyPrefixWith("replacekeyprefix");
rule.setRedirect(r);
RouteRuleCondition condition = new RouteRuleCondition();
condition.setHttpErrorCodeReturnedEquals("404");
condition.setKeyPrefixEquals("keyprefix");
rule.setCondition(condition);
config.getRouteRules().add(rule);

obsClient.setBucketWebsite("bucketname", config);
```

Configuring Redirection for All Requests

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

WebsiteConfiguration config = new WebsiteConfiguration();
RedirectAllRequest redirectAll = new RedirectAllRequest();
redirectAll.setHostName("www.example.com");
// Configure the support for both HTTP and HTTPS.
redirectAll.setRedirectProtocol(ProtocalEnum.HTTP);
config.setRedirectAllRequestsTo(redirectAll);

obsClient.setBucketWebsite("bucketname", config);
```

14.4 Viewing Website Hosting Settings

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getBucketWebsite** to view the hosting settings of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
```



```
WebsiteConfiguration config = obsClient.getBucketWebsite("bucketname");
Log.i("GetBucketWebsiteConfiguration", "\t" + config.getKey());
Log.i("GetBucketWebsiteConfiguration", "\t" + config.getSuffix());
for(RouteRule rule : config.getRouteRules()){
    Log.i("GetBucketWebsiteConfiguration", "\t" + rule);
}
```

14.5 Deleting Website Hosting Settings

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.deleteBucketWebsite** to delete the hosting settings of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.deleteBucketWebsite("bucketname");
```


15 Tag Management

15.1 Tagging Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Tags are used to identify and classify OBS buckets.

For more information, see **Tags**.

15.2 Setting Bucket Tags

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.setBucketTagging** to set bucket tags. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketTagInfo bucketTagInfo = new BucketTagInfo();

TagSet tagSet = new TagSet();

tagSet.addTag("tag1", "value1");
```



```
tagSet.addTag("tag2", "value2");

bucketTagInfo.setTagSet(tagSet);
obsClient.setBucketTagging("bucketname", bucketTagInfo);
```

NOTE

- A bucket can have up to 10 tags.
- The key and value of a tag can be composed of Unicode characters.

15.3 Viewing Bucket Tags

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getBucketTagging** to view bucket tags. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketTagInfo bucketTagInfo = obsClient.getBucketTagging("bucketname");
for(Tag tag : bucketTagInfo.getTagSet().getTags()){
    Log.i("GetBucketTagging", "\t" + tag.getKey() + ":" + tag.getValue());
}
```

15.4 Deleting Bucket Tags

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.deleteBucketTagging** to delete bucket tags. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.deleteBucketTagging("bucketname");
```


16 Event Notification

16.1 Event Notification Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

The event notification function allows users to be notified of their operations on buckets, ensuring users know events happened on buckets in a timely manner. Currently, OBS supports event notifications through Simple Message Notification (SMN) and FunctionGraph.

For more information, see **Event Notification**.

16.2 Setting Event Notification

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.setBucketNotification** to set event notification for a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketNotificationConfiguration bucketNotificationConfig = new BucketNotificationConfiguration();
```



```
TopicConfiguration config = new TopicConfiguration();
config.setId("id1");
config.setTopic("your topic");
config.getEventTypes().add(EventTypeEnum.ObjectCreatedAll);

Filter f = new Filter();
f.getFilterRules().add(new FilterRule("prefix", "smn"));
f.getFilterRules().add(new FilterRule("suffix", ".jpg"));
config.setFilter(f);
bucketNotificationConfig.addTopicConfiguration(config);

FunctionGraphConfiguration functionConfig = new FunctionGraphConfiguration();
functionConfig.setId("id2");
functionConfig.setFunctionGraph("your function");
functionConfig.getEventTypes().add(EventTypeEnum.OBJECT_CREATED_ALL);
Filter functionFilter = new Filter();
functionFilter.getFilterRules().add(new FilterRule("prefix", "function"));
functionFilter.getFilterRules().add(new FilterRule("suffix", ".mp4"));
functionConfig.setFilter(functionFilter);
bucketNotificationConfig.addFunctionGraphConfiguration(functionConfig);

obsClient.setBucketNotification("bucketname", bucketNotificationConfig);
```

16.3 Viewing Event Notification Settings

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

You can call **ObsClient.getBucketNotification** to view event notification settings of a bucket. Sample code is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketNotificationConfiguration config = obsClient.getBucketNotification("bucketname");

Log.i(config);
```

16.4 Disabling Event Notification

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

To disable event notification on buckets is to call **ObsClient.setBucketNotification** to clear all event notification settings. Sample code is as follows:


```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

obsClient.setBucketNotification("bucketname", new BucketNotificationConfiguration());
```


17 Server-Side Encryption

17.1 Server-Side Encryption Overview

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

OBS supports server-side encryption.

For more information, see **Server-Side Encryption**.

17.2 Encryption Description

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

The following table lists APIs related to server-side encryption:

Method in OBS Android SDK	Description	Supported Encryption Type
ObsClient.putObject	Sets the encryption algorithm and key during object upload to enable server-side encryption.	SSE-KMS SSE-C

Method in OBS Android SDK	Description	Supported Encryption Type
ObsClient.getObject	Sets the decryption algorithm and key during object download to decrypt the object.	SSE-C
ObsClient.copyObject	1. Sets the decryption algorithm and key for decrypting the source object during object copy. 2. Sets the encryption algorithm and key during object copy to enable the encryption algorithm for the target object.	SSE-KMS SSE-C
ObsClient.getObjectMetadata	Sets the decryption algorithm and key when obtaining the object metadata to decrypt the object.	SSE-C
ObsClient.initiateMultipartUpload	Sets the encryption algorithm and key when initializing a multipart upload to enable server-side encryption for the final object generated.	SSE-KMS SSE-C
ObsClient.uploadPart	Sets the encryption algorithm and key during multipart upload to enable server-side encryption for parts.	SSE-C
ObsClient.copyPart	1. Sets the decryption algorithm and key for decrypting the source object during multipart copy. 2. Sets the encryption algorithm and key during multipart copy to enable the encryption algorithm for the target part.	SSE-C

17.3 Example of Encryption

NOTICE

If you have any questions during development, post them on the **Issues** page of GitHub. For details about parameters and usage of each API, see the **API Reference**

Encrypting an Object to Be Uploaded

Sample code:


```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

PutObjectRequest request = new PutObjectRequest();
request.setBucketName("bucketname");
request.setObjectKey("objectname");
request.setFile(new File("localfile"));
// Set the SSE-C encryption algorithm.
SseCHeader sseCHeader = new SseCHeader();
sseCHeader.setAlgorithm(ServerAlgorithm.AES256);
sseCHeader.setSseCKeyBase64("your base64 sse-c key generated by AES-256 algorithm");

request.setSseCHeader(sseCHeader);
obsClient.putObject(request);
```

Decrypting a Downloaded Object

Sample code:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

GetObjectRequest request = new GetObjectRequest("bucketname", "objectname");
// Set the encryption algorithm to SSE-C.
SseCHeader sseCHeader = new SseCHeader();
sseCHeader.setAlgorithm(ServerAlgorithm.AES256);
// The key used here must be the one used for uploading the object.
sseCHeader.setSseCKeyBase64("your base64 sse-c key generated by AES-256 algorithm");

request.setSseCHeader(sseCHeader);
ObsObject obsObject = obsClient.getObject(request);

obsObject.getObjectContent().close();
```


18 Troubleshooting

18.1 OBS Server-Side Error Codes

If the OBS server encounters an error when processing a request, a response containing the error code and error description is returned. The following table lists details about each error code and HTTP status code.

Error Code	Description	HTTP Status Code
AccessDenied	Access denied.	403 Forbidden
AccessForbidden	Insufficient permission.	403 Forbidden
AccountProblem	Your account encounters a problem that prevents the operation from completing. The account may be expired or frozen.	403 Forbidden
AllAccessDisabled	You have no permission to perform the operation.	403 Forbidden
AmbiguousGrantByEmailAddress	Multiple accounts share one email address.	400 Bad Request
BadDigest	The specified value of **Content-MD5** does not match the value received by OBS.	400 Bad Request
BadDomainName	Invalid domain name.	400 Bad Request
BadRequest	Invalid request parameters.	400 Bad Request

Error Code	Description	HTTP Status Code
BucketAlreadyExists	The requested bucket name already exists. The bucket namespace is shared by all users of OBS. Try again with another bucket name.	409 Conflict
BucketAlreadyOwnedByYou	Your previous request for creating the named bucket succeeded and you already own it.	409 Conflict
BucketNotEmpty	The bucket that you tried to delete is not empty.	409 Conflict
CredentialsNotSupported	This request does not support security credentials.	400 Bad Request
CustomDomainAreadyExist	The configured domain already exists.	400 Bad Request
CustomDomainNotExist	The domain to be operated does not exist.	400 Bad Request
DeregisterUserId	The user has been deregistered.	403 Forbidden
EntityTooSmall	The size of the object to upload is smaller than the lower limit.	400 Bad Request
EntityTooLarge	The size of the object to upload is larger than the upper limit.	400 Bad Request
FrozenUserId	The user has been frozen.	403 Forbidden
IllegalVersioningConfigurationException	Invalid versioning configuration in the request.	400 Bad Request
IllegalLocationConstraintException	The configured region limitation is inconsistent with the region where it resides.	400 Bad Request
InArrearOrInsufficientBalance	The user has no permission to perform some operations due to being in arrears or insufficient balance.	403 Forbidden

Error Code	Description	HTTP Status Code
IncompleteBody	Incomplete request body.	400 Bad Request
IncorrectNumberOfFilesInPostRequest	Each POST request must contain one file to upload.	400 Bad Request
InlineDataTooLarge	The length of inline data exceeds the upper limit.	400 Bad Request
InsufficientStorageSpace	Insufficient storage space.	403 Forbidden
InternalError	Internal error. Please try again later.	500 Internal Server Error
InvalidAccessKeyId	The access key ID provided by the customer does not exist in the system.	403 Forbidden
InvalidAddressingHeader	The anonymous role must be specified.	N/A
InvalidArgument	Invalid parameter.	400 Bad Request
InvalidBucketName	The specified bucket name in the request is invalid.	400 Bad Request
InvalidBucket	The bucket to be accessed does not exist.	400 Bad Request
InvalidBucketState	Invalid bucket status.	409 Conflict
InvalidBucketStoragePolicy	An invalid new policy is specified during bucket policy modification.	400 Bad Request
InvalidDigest	The **Content-MD5** value specified in the HTTP header is invalid.	400 Bad Request
InvalidEncryptionAlgorithmError	Incorrect encryption algorithm.	400 Bad Request
InvalidLocationConstraint	The location specified during bucket creation is invalid.	400 Bad Request
InvalidPart	One or more specified parts are not found. The parts may not be uploaded or the specified entity tags (ETags) do not match the parts' ETags.	400 Bad Request

Error Code	Description	HTTP Status Code
InvalidPartOrder	Parts are not listed in ascending order by part number.	400 Bad Request
InvalidPayer	All access requests for this object are invalid.	403 Forbidden
InvalidPolicyDocument	The content of the form does not meet the conditions specified in the policy document.	400 Bad Request
InvalidRange	The requested range cannot be obtained.	416 Client Requested Range Not Satisfiable
InvalidRedirectLocation	Invalid redirect location.	400 Bad Request
InvalidRequest	Invalid request.	400 Bad Request
InvalidRequestBody	Invalid POST request body.	400 Bad Request
InvalidSecurity	Invalid security credentials.	403 Forbidden
InvalidStorageClass	Invalid storage class.	400 Bad Request
InvalidTargetBucketForLogging	The delivery group has no ACL permission for the target bucket.	400 Bad Request
InvalidURI	The specified URI cannot be resolved.	400 Bad Request
KeyTooLong	The provided key is too long.	400 Bad Request
MalformedACLError	The XML format is incorrect or does not meet the format requirements.	400 Bad Request
MalformedError	The XML format in the request is incorrect.	400 Bad Request
MalformedLoggingStatus	The XML format of **Logging** is incorrect.	400 Bad Request
MalformedPolicy	The bucket policy does not pass.	400 Bad Request

Error Code	Description	HTTP Status Code
MalformedPOSTRequest	The body of the POST request is in an incorrect format.	400 Bad Request
MalformedQuotaError	The Quota XML format is incorrect.	400 Bad Request
MalformedXML	This error code is returned after you send an XML file in incorrect format. The error message states: "The XML you provided was not well-formed or did not validate against our published schema."	400 Bad Request
MaxMessageLengthExceeded	The request is too long.	400 Bad Request
MaxPostPreDataLengthExceeded Error	The POST request fields prior to the file to upload are too large.	400 Bad Request
MetadataTooLarge	The size of the metadata header exceeds the upper limit.	400 Bad Request
MethodNotAllowed	The specified method is not allowed against the requested resource.	405 Method Not Allowed
MissingContentLength	The HTTP header **Content-Length** is not provided.	411 Length Required
MissingRegion	The region information is missing in the request, and the default region is required in the system.	400 Bad Request
MissingRequestBodyError	This error code is returned after you send an empty XML file. The error message states: "Request body is empty."	400 Bad Request
MissingRequiredHeader	Required headers are missing in the request.	400 Bad Request
MissingSecurityHeader	A required header is missing in the request.	400 Bad Request
NoSuchBucket	The bucket does not exist.	404 Not Found

Error Code	Description	HTTP Status Code
NoSuchBucketPolicy	No bucket policy exists.	404 Not Found
NoSuchCORSConfiguration	No CORS configuration exists.	404 Not Found
NoSuchCustomDomain	The requested user domain does not exist.	404 Not Found
NoSuchKey	The key does not exist.	404 Not Found
NoSuchLifecycleConfiguration	The requested lifecycle rule does not exist.	404 Not Found
NoSuchPolicy	The policy name does not exist.	404 Not Found
NoSuchUpload	The multipart upload does not exist. The upload ID does not exist or the multipart upload has been aborted or completed.	404 Not Found
NoSuchVersion	The specified version ID does not match any existing version.	404 Not Found
NoSuchWebsiteConfiguration	The requested website does not exist.	404 Not Found
NotImplemented	The provided header implies a function that is unavailable.	501 Not Implemented
NotSignedUp	Your account is not signed up for OBS. OBS is available only after you sign up.	403 Forbidden
OperationAborted	A conflicting operation is being performed on this resource. Try again later.	409 Conflict
PermanentRedirect	The requested bucket must be addressed using a specified endpoint. Send all future requests to this endpoint.	301 Moved Permanently
PreconditionFailed	At least one of the specified preconditions is not met.	412 Precondition Failed
Redirect	The request is temporarily redirected.	307 Moved Temporarily

Error Code	Description	HTTP Status Code
RequestIsNotMultiPartContent	A bucket POST request must contain an enclosure-type multipart or the form-data.	400 Bad Request
RequestTimeout	No read or write operation has been performed within the timeout period of the socket connection between the user and the server.	400 Bad Request
RequestTimeTooSkewed	The request time and the server's time differ a lot.	403 Forbidden
RequestTorrentOfBucketError	Requesting the bucket's torrent file is not allowed.	400 Bad Request
ServiceNotImplemented	The request method is not implemented by the server.	501 Not Implemented
ServiceNotSupported	The request method is not supported by the server.	409 Conflict
ServiceUnavailable	The server is overloaded or has internal errors.	503 Service Unavailable
SignatureDoesNotMatch	The provided signature does not match the signature calculated by OBS. Check your AK and SK and signature calculation method.	403 Forbidden
SlowDown	Too frequent requests. Reduce your request frequency.	503 Service Unavailable
System Capacity Not enough	Insufficient system space.	403 Forbidden
TooManyCustomDomains	Too many user domains are configured.	400 Bad Request
TemporaryRedirect	The request is redirected to the bucket while the domain name server (DNS) is being updated.	307 Moved Temporarily
TooManyBuckets	You have attempted to create more buckets than allowed.	400 Bad Request

Error Code	Description	HTTP Status Code
TooManyObjectCopied	The number of copies of a single object exceeds the upper limit.	400 Bad Request
TooManyWrongSignature	The request is rejected due to high-frequency errors.	400 Bad Request
UnexpectedContent	This request does not support fields with content.	400 Bad Request
UnresolvableGrantByEmailAddress	The provided email address does not match any recorded accounts.	400 Bad Request
UserKeyMustBeSpecified	The user's AK is not carried in the request.	400 Bad Request
WebsiteRedirect	The website request lacks **bucketName**.	301 Moved Permanently
KMS.DisabledException	The master key is disabled in SSE-KMS mode.	400 Bad Request
KMS.NotFoundException	The customer master key (CMK) does not exist in SSE-KMS mode.	400 Bad Request
RestoreAlreadyInProgress	The Archive object is being restored. The request conflicts with another one.	409 Conflict
ObjectHasAlreadyRestored	The objects have been restored and the retention period of the objects cannot be shortened.	409 Conflict
InvalidObjectState	The restored object is not an Archive object.	403 Forbidden
InvalidTagError	An invalid tag is provided when configuring the bucket tag.	400 Bad Request
NoSuchTagSet	The specified bucket does not have a tag.	404 Not Found

18.2 SDK Custom Exceptions

SDK custom exceptions (**ObsException**), thrown by **ObsClient**, are inherited from class **java.lang.RuntimeException**. Exceptions are usually OBS server errors,

including **OBS error codes** and error information. This facilitates users to locate problems and troubleshot faults.

ObsException contains the following error information:

- **ObsException.getResponseCode**: HTTP status code
- **ObsException.getErrorCode**: Error code returned by the OBS server
- **ObsException.getErrorMessage**: Error description returned by the OBS server
- **ObsException.getErrorRequestId**: Request ID returned by the OBS server
- **ObsException.getErrorHostId**: Requested server ID
- **ObsException.getResponseHeaders:** HTTP response headers

18.3 SDK Common Response Headers

After you call an API in an instance of **ObsClient**, an instance of the **HeaderResponse** class (or of its sub-class) will be returned. It contains information about HTTP/HTTPS response headers.

Sample code for processing public response headers:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
// Create an instance of ObsClient.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);
HeaderResponse response = obsClient.createBucket("bucketname");

// Obtain the UUID from the public response headers.
Log.i("CreateBucket", "\t" + response.getRequestId());

obsClient.close();
```

18.4 Log Analysis

Log Path

The log path of the OBS Android SDK is specified using **LogConfigurator.setLogFileDir**. Logs are saved in the **logs** directory of the SD card by default. The log file name is **OBS-SDK.log**.

Log Format

The SDK log format is: Log time|thread ID|log level|log content. The following are examples:

```
2017-08-21 17:40:07 133|main|INFO |HttpClient cost 157 ms to apply http request
2017-08-21 17:40:07 133|main|INFO |Received expected response code: true
2017-08-21 17:40:07 133|main|INFO |expected code(s): [200, 204].

2017-08-21 17:40:06 820|main|INFO |Storage|1|HTTP+XML|ObsClient||||2017-08-21 17:40:05|2017-08-21 17:40:06|||0|
2017-08-21 17:40:07 136|main|INFO |Storage|1|HTTP+XML|setObjectAcl||||2017-08-21 17:40:06|2017-08-21 17:40:07|||0|
2017-08-21 17:40:07 137|main|INFO |ObsClient [setObjectAcl] cost 312 ms
```


Log Level

When current logs cannot be used to troubleshoot system faults, you can change the log level to obtain more information. You can obtain the most information in **TRACE** logs and the least information in **ERROR** logs. You can call **LogConfigurator.setLogLevel** to set bucket tags.

The following table describes each log level in detail.

Log Level	Description	Value in OBS Android SDK
OFF	Close level. If this level is set, logging will be disabled.	LogConfigurator.OFF
TRACE	Trace level. If this level is set, all log information will be printed. This level is not recommended.	LogConfigurator.TRACE
DEBUG	Debugging level. If this level is set, information about logs of the **INFO** level and above, HTTP/HTTPS request and response headers, and **StringToSign** information calculated by authentication algorithm will be printed.	LogConfigurator.DEBUG
INFO	Information level. If this level is set, information about logs of the **WARN** level and above, time consumed for each HTTP/HTTPS request, and time consumed for calling the **ObsClient** API will be printed.	LogConfigurator.INFO
WARN	Warning level. If this level is set, information about logs of the **ERROR** level and above, as well as information about some critical events (for example, the number of retry attempts exceeds the upper limit) will be printed.	LogConfigurator.WARN
ERROR	Error level. If this level is set, only error information will be printed.	LogConfigurator.ERROR

Log File Cannot Be Generated

If logs cannot be generated, check whether the following permissions are added in **AndroidManifest.xml**:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

18.5 Lack of Classes

If an error indicating the lack of a class is displayed when you are using OBS Android SDK for secondary development, ensure that the following dependent

libraries are saved in the **libs** directory, besides JAR packages named **esdk-obs-android-*x.x.x*.jar**:

- java-xmlbuilder-1.1.jar
- okhttp-3.10.0.jar
- okio-1.14.0.jar

18.6 Connection Timeout

If the error code obtained through **ObsException.getResponseCode** is **408**, connection to OBS timed out. Such an error is often caused by an incorrect endpoint or network disconnection. Check whether the endpoint is correct or whether the network connection is normal.

18.7 Resources Cannot Be Released

If memory leakages occur or OBS Android SDK is not disconnected from the OBS server after you use OBS Android SDK, check whether **ObsClient.close** and **ObsObject.getObjectContent.close** are properly called to release resources.

18.8 Unmatched Signatures

If the HTTP status code obtained from **ObsException** is **403** and the OBS server-side error code is **SignatureDoesNotMatch**, check whether the AK/SK is correct.

19 FAQ

19.1 How Can I Perform a Multipart Upload?

For details, see **Performing a Multipart Upload**.

19.2 How Can I Create a Folder?

For details, see **Creating a Folder**.

19.3 How Can I List All Objects in a Bucket?

For details, see **Listing Objects** and **Listing Versioning Objects**.

19.4 How Can I Use a URL for Authorized Access?

See **Using a Temporary URL for Authorized Access**.

19.5 How Can I Upload an Object in Browser-Based Mode?

For details, see **Performing a Browser-Based Upload**.

19.6 How Can I Download a Large Object in Multipart Mode?

For details, see **Performing a Partial Download**.

19.7 How Can I Set an Object to Be Accessible to Anonymous Users?

For details, see **Website File Hosting**.

19.8 How Can I Identify the Endpoint and Location of OBS?

For details, see **Obtaining Endpoints**.

19.9 How Can I Obtain the AK/SK?

For details, see **Setting Up an OBS Environment**.

19.10 Does the SDK Support Uploading, Downloading, or Copying Objects in a Batch?

No.

Currently, the SDK does not provide such APIs. You need to encapsulate the service codes for uploading, downloading, or copying objects in a batch by yourself. The procedure is as follows:

Step 1 Invoke **listObjects** to list all objects to be uploaded, downloaded, or copied. For details about the code example, see **Listing Objects**.

Step 2 Invokes the API for uploading, downloading, or copying a single object for the listed objects.

----End

The example code for uploading objects in a batch is as follows:

```
String endPoint = "https://your-endpoint";
String ak = "*** Provide your Access Key ***";
String sk = "*** Provide your Secret Key ***";
final String bucketName = "bucketname";
// Define the prefix of objects in a bucket.
final String objectPre = "object/";
// Folder to be uploaded
final String localDirPath = "localDirPath";
final List<File> list = new ArrayList<>();
// Scan all objects in the folder.
static void listFiles(File file){
    File[] fs = file.listFiles();
    assert fs != null;
    if (fs.length < 1){
        // If uploading an empty folder is required, add it to the list.
        list.add(file);
    }else{
        for (File f:fs){
            if (f.isDirectory()){
                listFiles(f);
```



```
            }
            if (f.isFile()){
                // Add objects to be uploaded to the list.
                list.add(f);
            }
        }
    }
}
// Traverse the folder to be uploaded and obtain all objects to be uploaded.
File file = new File(localDirPath);
listFiles(file);

// Create an instance of ObsClient.
final ObsClient obsClient = new ObsClient(ak, sk, endPoint);

// Initialize the thread pool.
ExecutorService executorService = Executors.newFixedThreadPool(20);

// Concurrently upload parts.
for (File f:list){
    executorService.execute(() -> {
        if (f.isDirectory()){
            // For empty folders, create empty folder objects in the bucket.
            String remoteObjectKey = objectPre + f.getPath().substring(localDirPath.length() + 1) + "/";
            obsClient.putObject(bucketName, remoteObjectKey, new ByteArrayInputStream(new byte[0]));
        }else{
            String remoteObjectKey = objectPre + f.getPath().substring(localDirPath.length() + 1);
            obsClient.putObject(bucketName, remoteObjectKey, new File(f.getPath()));
        }
    });
}

// Wait until the upload is complete.
executorService.shutdown();
while (!executorService.isTerminated())
{
    try
    {
        executorService.awaitTermination(5, TimeUnit.SECONDS);
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}
// Close obsClient.
try {
    obsClient.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

NOTE

You can use multiple threads to concurrently upload, download, and copy data to improve efficiency.

19.11 What Should I Do If the HTTP Status Code 405 Is Reported?

If an API calling fails and the HTTP status code returned is 405, check whether the region supports the called API.

A API Reference

For details about all parameters and definitions of APIs in the OBS Android SDK, see the **Object Storage Service Android SDK API Reference**.

B Change History

Date	What's New
2020-04-27	This is the thirteenth official release. • Version 3.20.3
2020-02-20	This is the twelfth official release. • Version 3.20.1
2019-11-20	This is the eleventh official release. Reorganized the positions of the following sections: • Example Programs • Technical Support Channels
2019-08-10	This is the tenth official release. Modified the following section: **Creating an Instance of ObsClient**
2019-03-30	This is the ninth official release. Added the following section: **API Reference**
2019-03-05	This is the eighth official release. Modified the following section: Added the description and code example for the function workflow service in each section of "Event Notification". Added the following section: **Does the SDK Support Uploading, Downloading, or Copying Objects in a Batch?**
2019-02-18	This is the seventh official release. Modified the following section: Modified content in section **Log Analysis**.

Date	What's New
2018-10-31	This is the sixth official release. Added the following sections: **Obtaining Upload Progresses** **Obtaining Download Progresses** Modified the following section: Added some configuration parameters in **Configuring an Instance of ObsClient**.
2018-08-31	This is the fifth official release. Added the following section: **Setting Object Properties** **Performing an Appendable Upload** **Performing a Browser-Based Upload**
2018-01-31	This is the fourth official release. Updated the following sections: Added **Setting Up an OBS Environment**, **Preparing a Development Environment**, **Installing the SDK**, and **Obtaining Endpoints** in the "Quick Start" chapter. Added **Example Programs** and removed the content about setting up an OBS environment in the "Related Resources" chapter. Removed content about endpoints from the "Initialization" chapter. Modified the following sections: Modified sample code in "Temporary Authentication." Modified the sample code in "Setting Lifecycle Rules." Modified the sample code in "Viewing Lifecycle Rules."
2017-12-31	This is the third official release. Added the following section: **Processing an Image** Added the object transition policy feature in **Lifecycle Management**.
2017-11-30	This is the second official release. Added the following sections: **Storage Class** **Performing a Resumable Upload** **Performing a Resumable Download** Modified the following section: Added "Setting the Storage Class for an Object" in **Setting Object Properties**.

Date	What's New
2017-10-31	This is the first official release.

