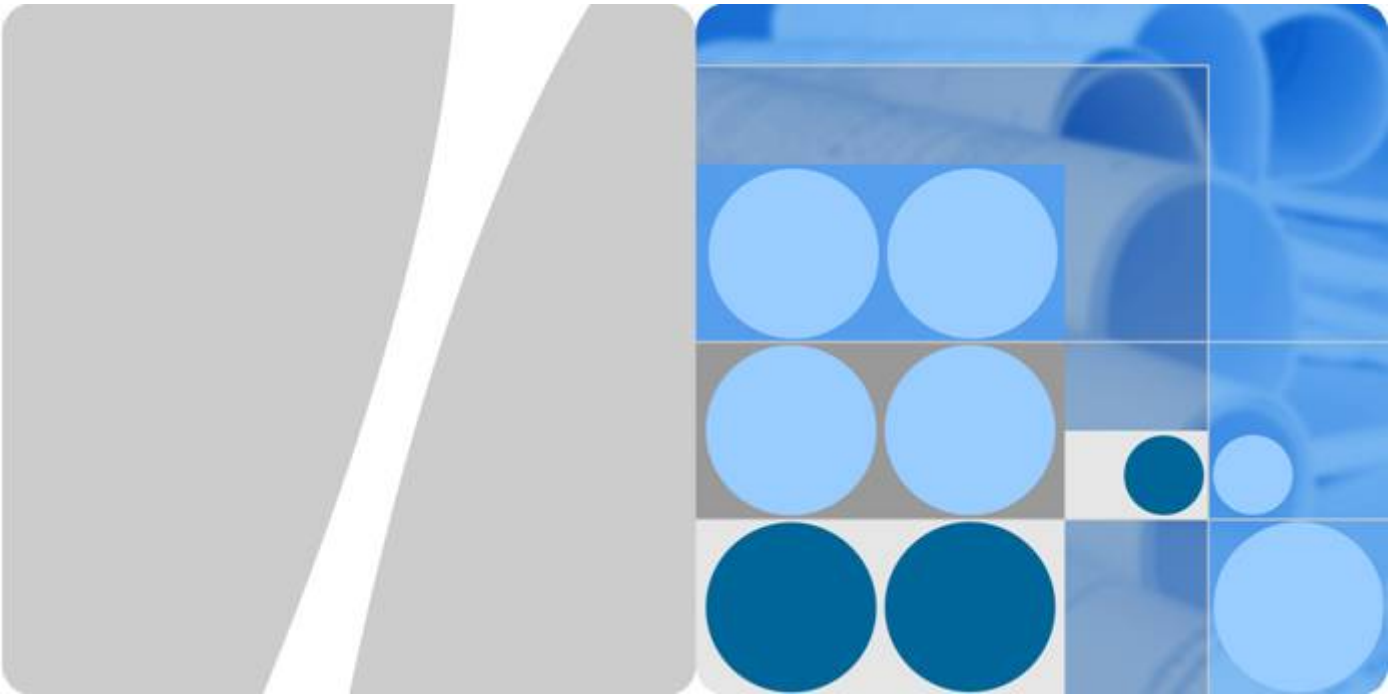

Object Storage Service

Python SDK Developer Guide

Issue 12

Date 2019-09-05

HUAWEI TECHNOLOGIES CO., LTD.

HUAWEI

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

SDK Source Codes and API Documentation

- Latest version of OBS Python SDK source code: Click **here** to download.
- Earlier version of OBS Python SDK: Click **here** to download.
- OBS Python SDK API document: **OBS Python SDK API Reference**

Compatibility

- For details about the change history of versions, see **ChangeLog**.
- Recommended Python 2.*x* version: 2.7.*x*
- Recommended Python 3.*x* versions: 3.3, 3.4, 3.5, and 3.6
- Namespace: This version is incompatible with the earlier version (2.1.*x*). All modules are saved in the **obs** package.
- API functions: Compatible with earlier versions (2.1.*x*).

2 Example Programs

OBS Python SDK provides abundant example programs for user reference and direct use. These programs can be obtained from the OBS Python SDK development package. For example, files in **eSDK_Storage_OBS_*<versionId>*_Python/examples** obtained by decompressing **eSDK_Storage_OBS_*<versionId>*_Python.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	Description
bucket_operations_sample	How to use bucket-related APIs.
object_operations_sample	How to use object-related APIs.
download_sample	How to download an object.
create_folder_sample	How to create a folder.
delete_objects_sample	How to delete objects in a batch.
list_objects_sample	How to list objects.
list_versions_sample	How to list versioning objects.
list_objects_in_folder_sample	How to list objects in a folder.
object_meta_sample	How to customize object metadata.
simple_multipart_upload_sample	How to perform a multipart upload.

Sample Code	Description
restore_object_sample	How to download Archive objects.
concurrent_copy_part_sample	How to concurrently copy parts of a large object.
concurrent_download_object_sample	How to concurrently download parts of a large object.
concurrent_upload_part_sample	How to concurrently upload parts of a large object.
post_object_sample	How to perform a browser-based upload.
temporary_signature_sample	How to use URLs for authorized access.

3 Quick Start

3.1 Before You Start

- Ensure that you are familiar with OBS basic concepts, such as **bucket**, **object**, and **AK and SK**.
- You can read **General Examples of ObsClient** to understand how to call OBS Python SDK APIs in a general manner.
- Each time an API is called by using ObsClient (without any exception thrown), an **SDK common result object** will be returned. You can obtain the HTTP status code from the result to check whether the operation is successful.
- 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

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 the management 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 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.
- A secret access key (SK) is the key used by users to access OBS. It is the authentication information generated based on the AK and the request header. An SK matches an AK, and they group into a pair.

Access keys are classified into permanent access keys (AK/SK) and temporary access keys (AK/SK and security token). Permanent access keys are valid for a year after creation. Each user can create up to two valid AK/SK pairs. Temporary access keys can be used to access OBS only within the specified validity period. After the temporary access keys expire, they need to be obtained again. For security purposes, you are advised to use temporary access keys to access OBS, or periodically update your access keys if you use permanent access keys. The following describes how to obtain access keys of these two types.

- Permanent access keys:
  a. Log in to OBS Console.
  b. In the upper right corner of the page, hover the cursor over the username and click **My Credentials**.
  c. On the **My Credentials** page, select **Access Keys** in the navigation pane on the left.
  d. On the **Access Keys** page, click **Create Access Key**.
  e. 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.

  f. Click **OK**.
  g. In the **Download Access Key** dialog box that is displayed, click **OK** to save the access keys to your browser's default download path.
  h. Open the downloaded **credentials.csv** file to obtain the access keys (AK and SK).

     NOTE

     - A user can create a maximum of 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.

- Temporary access keys:

  The temporary AK/SK and security token are temporary access tokens issued by the system to users. The validity period ranges from 15 minutes to 24 hours which can be set using APIs. After the validity period expires, users need to obtain the access keys again. The temporary AK/SK and security token shall observe the principle of least privilege. When the temporary AK/SK are used for authentication, the temporary AK/SK and security token must be used at the same time.

  For details about how to obtain temporary access keys, see **Obtaining a Temporary AK/SK**.

  For details about how to use temporary access keys, see **Creating an Instance of ObsClient**.

----End

3.3 Preparing a Development Environment

- Download Python of a proper version from **Python's official website** and install it.
- Download the latest community version of PyCharm from **PyCharm's official website**.

3.4 Installing the SDK

(Recommended) Installing the SDK Using pip

Step 1 Run the **pip -V** command to check the pip version and ensure that pip is installed.

Step 2 Run the **pip install esdk-obs-python --trusted-host pypi.org** command to start the installation.

----End

NOTE

- If pip has not been installed, install it by referring to the **pip official website**.
- On a Windows operating system, the message "Not internal or external command" is displayed when you run the pip command. In this case, add the pip installation directory (generally the Scripts folder in the directory where the Python is located) to the Path environment variable.
- You may need to restart the computer for the environment variables to take effect.

Installing the SDK Using Source Codes

The following procedures use installing the latest version of OBS Python SDK as an example.

Method 1: Direct installation

Step 1 Download the **OBS Python SDK software package**.

Step 2 Decompress the development package to obtain folder **src** (SDK source code), folder **examples** (sample code), file **README.txt** (feature description file of SDK versions), and file **log.conf** (SDK log configuration file).

Step 3 Use PyCharm to create a project, copy the folders and files obtained in the previous step to the project, right-click folder **src**, and choose **Mark Directory as** > **Sources Root**.

NOTE

After the configuration, the directory structure is similar to the following:

```
├── examples
├── src
├── log.conf
└── README.txt
```

----End

Method 2: Using the setuptools Tool

Step 1 Download the OBS Python SDK development package and decompress it.

Step 2 Download and install **setuptools**.

Step 3 On the command-line interface (CLI), go to folder **src** under the directory where the development package is decompressed.

Step 4 Run the **python setup.py install** command to install the SDK.

Step 5 After the installation, check whether a folder named **esdk_obs_python-<versionId>-*.egg** is generated in **Lib/site-package** under the Python installation directory.

NOTE

- If you use this method to install the SDK, you need to delete folder **esdk_obs_python-*<versionId>*-*.egg** when you re-install the SDK.
- If SDK modules cannot be loaded after you have performed the previous steps, you can directly add absolute path of the **src** directory in OBS Python SDK to the **sys.path** list.

----End

3.5 Obtaining Endpoints

- 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

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
# Use the instance to access OBS.

# Close obsClient.
obsClient.close()
```

NOTE

For more information, see chapter "Initialization."

3.7 Creating a Bucket

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:

```
# Invoke createBucket to create a bucket.
resp = obsClient.createBucket('bucketname')
if resp.status < 300:
    # Return the request ID.
    print('requestId:', resp.requestId)
else:
    # Return the error code.
    print('errorCode:', resp.errorCode)
    # Return error information.
    print('errorMessage:', resp.errorMessage)
```

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-like address.
  - 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 **6.1 Creating a Bucket**.

NOTICE

When creating a bucket, you do not need to specify the region if the endpoint belongs to the default region (cn-north-1). If the endpoint belongs to a region other than the default one, you need to specify the region to which the endpoint belongs. For more information, refer to **Creating a Bucket with Parameters Specified**. Click **here** to query currently valid regions.

3.8 Uploading an Object

Sample code:

```
# Invoke the putContent interface to upload the object to the bucket.
resp = obsClient.putContent('bucketname', 'objectname', 'Hello OBS')
if resp.status < 300:
    # Return the request ID.
    print('requestId:', resp.requestId)
else:
    # Return the error code.
    print('errorCode:', resp.errorCode)
    # Return error information.
    print('errorMessage:', resp.errorMessage)
```

NOTE

For more information, see **7.1 Object Upload Overview**.

3.9 Downloading an Object

Sample code:

```
# Invoke the getObject interface to download objects in the bucket.
resp = obsClient.getObject('bucketname', 'objectname')
if resp.status < 300:
    # Return the request ID.
    print('requestId:', resp.requestId)
    # Read the object content in the response body.
    if resp.body and resp.body.response:
        while True:
            chunk = resp.body.response.read(65536)
            if not chunk:
                break
            print(chunk)
        resp.body.response.close()
else:
    # Return the error code.
    print('errorCode:', resp.errorCode)
    # Return error information.
    print('errorMessage:', resp.errorMessage)
```

NOTE

For more information, see **8.1 Object Download Overview**.

3.10 Listing Objects

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


```
# Invoke the listObjects interface to list all objects in a specified bucket.
resp = obsClient.listObjects('bucketname')
if resp.status < 300:
    # Return the request ID.
    print('requestId:', resp.requestId)
    index = 0
    # Return information about all objects.
    for content in resp.body.contents:
        print('content [' + str(index) + ']')
        # Return the object name.
        print('key:', content.key)
        # Return the last modification time of the object.
        print('lastModified:', content.lastModified)
        # Return the object size.
        print('size:', content.size)
        index += 1
else:
    # Return the error code.
    print('errorCode:', resp.errorCode)
    # Return error information.
    print('errorMessage:', resp.errorMessage)
```

NOTE

- In the previous sample code, 1000 objects will be listed, by default.
- For more information, see **Listing Objects**.

3.11 Deleting an Object

Sample code:

```
# Invoke the deleteObject interface to delete a specified object in a specified bucket.
resp = obsClient.deleteObject('bucketname', 'objectname')
if resp.status < 300:
    # Return the request ID.
    print('requestId:', resp.requestId)
else:
    # Return the error code.
    print('errorCode:', resp.errorCode)
    # Return error information.
    print('errorMessage:', resp.errorMessage)
```

3.12 General Examples of ObsClient

Each time an API is called by using **ObsClient** (without any exception thrown), an **SDK common result object** will be returned. You can obtain the HTTP status code from this object to check whether the operation is successful.

Sample code:

```
# Import the module.
from obs import ObsClient

# You can reserve only one global instance of ObsClient in your project.
# ObsClient is thread-safe and can be simultaneously used by multiple threads.
# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

try:
```



```
    # Call an API to perform related operations, for example, uploading an object.
    resp = obsClient.putFile('bucketname', 'objectname', 'localfile')
    # If no exception occurs and the API call is complete, check the HTTP status code.
    if resp.status < 300:
        # The operation is successful
        print('requestId:', resp.requestId)
        # Process the business logic after the operation is successful.
    else:
        # If the operation fails, obtain the exception details.
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        print('requestId:', resp.requestId)
        print('hostId:', resp.hostId)
except Exception as e:
    import traceback
    # If any error occurs, print the exception stacks.
    print(traceback.format_exc())
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.
    obsClient.close()
```

3.13 Pre-defined Constants

OBS Python SDK provides a group of pre-defined constants that can be directly used. For details, see the *OBS Python SDK API Reference*.

4 Initialization

4.1 Configuring the AK and SK

To use OBS, you need a valid pair of AK and SK for signature authentication. For details, see **3.2 Setting Up an OBS Environment**.

After obtaining the AK and SK, you can start initialization.

4.2 Creating an Instance of ObsClient

ObsClient functions as the Python client for accessing OBS. It offers users a series of APIs for interaction with OBS. These APIs are used for managing and operating resources, such as buckets and objects, stored in OBS. To use OBS Python SDK to send a request to OBS, you need to initialize an instance of ObsClient and modify parameters related to initial configurations of the instance based on actual needs.

- You can create an instance of ObsClient by using a constructor function. Sample code for creating an instance of ObsClient using permanent access keys (AK/SK):

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-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):

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
```



```
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    security_token='*** Provide your Security Token ***',
    server='https://your-endpoint'
)

# Use the instance to access OBS.

# Close ObsClient.
obsClient.close()
```

- You can also create an instance of ObsClient by using temporary access keys obtained by configuring system environment variables or by accessing an ECS.
  - Sample code for creating an instance of ObsClient using ENV:

    ```
    # Import the module.
    from obs import ObsClient
    from obs import loadtoken

    # Create an instance of ObsClient.
    # Provide ENV to obtain the access keys.
    obsClient = ObsClient(
        server='https://your-endpoint',
        security_provider_policy='ENV'
    )

    # Use the instance to access OBS.

    # Close ObsClient.
    obsClient.close()
    ```

    NOTE

    In the preceding method, access keys are searched from the environment variables of the current system. The **OBS_ACCESS_KEY_ID** and **OBS_SECRET_ACCESS_KEY** fields need to be defined in the corresponding environment variables. If temporary access keys are used, the **OBS_SECURITY_TOKEN** field must also be defined in the environment variables.

  - Sample code for creating an instance of ObsClient using ECS:

    ```
    # Import the module.
    from obs import ObsClient
    from obs import loadtoken

    # Create an instance of ObsClient.
    # Provide ECS to obtain the temporary access keys.
    obsClient = ObsClient(
        server='https://your-endpoint',
        security_provider_policy='ECS'
    )

    # Use the instance to access OBS.

    # Close ObsClient.
    obsClient.close()
    ```

    NOTE

    When an application is deployed on an ECS, temporary access keys can be obtained automatically using the preceding methods and updated periodically.

NOTICE

When obtaining temporary access keys using this method, ensure that the UTC time of the server is the same as that of the environment where the application is deployed. Otherwise, the temporary access keys may fail to be updated in time.

- In addition to the preceding methods, you can also search in sequence to obtain the corresponding access keys from the environment variables and ECSs.
  - You can set **security_provider_policy** to **OBS_DEFAULT** to specify that ObsClient searches for access keys in sequence.

```
# Import the module.
from obs import ObsClient
from obs import loadtoken

# Create an instance of ObsClient.
# Search for access keys from environment variables and ECSs in sequence.
obsClient = ObsClient(
    server='https://your-endpoint',
    security_provider_policy='OBS_DEFAULT'
)

# Use the instance to access OBS.

# Close ObsClient.
obsClient.close()
```

NOTE

In the preceding method, **security_provider_policy** is set to **OBS_DEFAULT**, which specifies that ObsClient searches for access keys in sequence from the predefined list. By default, the system provides two predefined search methods: obtaining the access keys from the environment variables and obtaining from ECSs. ObsClient searches for the access keys from the environment variables first and then from ECSs. In this case, ObsClient is created using the first pair of access keys obtained in the search.

NOTE

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

4.3 Creating an Instance of BucketClient

You can use **ObsClient** to create an instance of **BucketClient** which allows you to call APIs without **bucketName**. Sample code is as follows:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Create an instance of BucketClient.
bucketClient = obsClient.bucketClient('bucketname')
# Create a bucket.
```



```
resp = bucketClient.createBucket()
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

Except **ObsClient.listBuckets**, **ObsClient.createSignedUrl**, and **ObsClient.createPostSignature**, **BucketClient** can implement the same APIs as **ObsClient**, without requiring the **bucketName** parameter.

4.4 Configuring an Instance of ObsClient

You can set the following initialization parameters to configure an instance of ObsClient:

Parameter	Description	Recommended Value
access_key_id	AK The default value is an empty string, indicating an anonymous user.	N/A
secret_access_key	SK The default value is an empty string, indicating an anonymous user.	N/A
security_token	Security token in the temporary access keys	N/A
server	Endpoint for accessing OBS, which can contain the protocol type, domain name, and port number. For example, **https://your-endpoint:443**.	N/A
max_retry_count	Maximum number of retries when an HTTP/HTTPS connection is abnormal. The default value is **3**.	[1, 5]
max_redirect_count	Maximum number of times that the HTTP/HTTPS request is redirected. The default value is **10**.	[1, 10]
timeout	Timeout period (in seconds) of an HTTP/HTTPS request. The default value is **60**.	[10, 60]

Parameter	Description	Recommended Value
ssl_verify	Whether to verify server-side certificates. Possible values are: • Path to the server-side root certificate file in PEM format • **True**: The certificate list will be obtained from the root certificate library and the certificates of the operating system (Windows only) will be verified. • **False**: The server-side certificates will not be verified. The default value is **False**.	N/A
chunk_size	Block size for reading and writing socket streams. The default value is **65536**.	Default
long_conn_mode	Whether to enable the persistent connection mode. The default value is **False**.	N/A
proxy_host	Host address of the proxy server. This value is **None** by default.	N/A
proxy_port	Port number of the proxy server. This value is **None** by default.	N/A
proxy_username	User name used for connecting to the proxy server. This value is **None** by default.	N/A
proxy_password	Password used for connecting to the proxy server. This value is **None** by default.	N/A
is_cname	Whether to use self-defined domain name to access OBS. The default value is **False**.	N/A
security_providers	Specifies the allowed access key search methods. The default value is **None**. **NOTE** The value of **security_providers** must be in a list. The default value **None** indicates the default search methods to obtain the access keys from the environment variables and from ECSs. If this parameter is specified, the default search methods are not provided. Instead, the search methods specified by **security_providers** are used.	N/A

Parameter	Description	Recommended Value
security_provider_policy	Specifies the allowed access key search policy. The default value is **None**. **NOTE** • This parameter is used to set the search policy. The default value **None** indicates the specified access keys are displayed. In addition, if the access key parameters are specified, **security_provider_policy** is ignored. • If **security_provider_policy** is set to **OBS_DEFAULT**, the access keys are obtained by searching in sequence. • If **security_provider_policy** is set to the predefined methods (ENV or ECS), the access keys are obtained using the corresponding method.	N/A

NOTE

- Parameters whose recommended value is **N/A** need to be set according to the actual conditions.
- If the network is unstable, you are advised to set a larger value for **timeout**.
- If the value of **server** does not contain any protocol, HTTPS is used by default.

NOTICE

- If the persistent connection mode is enabled, you must call **ObsClient.close** to close ObsClient explicitly to reclaim connection resources.
- For the sake of high DNS resolution performance and OBS reliability, you can set **server** only to the domain name of OBS, instead of the IP address.

4.5 Configuring SDK Logging

OBS Python SDK provides the logging function. To enable this function, you can call the relevant API function to specify the path to the log configuration file. The procedure is as follows:

Step 1 Find the **log.conf** file in the OBS Python SDK.

Step 2 Modify parameters in **log.conf** as needed.

Step 3 Call **ObsClient.initLog** to enable the logging function. Sample code is as follows:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
```



```
    server='https://your-endpoint'
)

# Import the log module.
from obs import LogConf

# Specify the path to the log configuration file and initialize logs of ObsClient.
obsClient.initLog(LogConf('./log.conf'), 'obsclient');

# Use ObsClient to access OBS.

# Disable ObsClient logging.
obsClient.close()
```

----End

NOTE

- The logging function is disabled by default. You need to enable it if needed.
- For details about SDK logs, see **Log Analysis**.

NOTICE

The log module of the OBS Python SDK is thread safe but not process safe. If ObsClient is used in multi-process scenarios, you must configure an independent log path for each instance of ObsClient to prevent conflicts when multiple processes write logs concurrently.

4.6 Configuring Server-Side Certificate Verification

OBS Python SDK supports server-side certificate verification to ensure that OBS is provided by trusted servers. To configure server-side verification, perform the following:

Step 1 Obtain the root certificate of the OBS server (for example, open Internet Explorer and choose **Internet Options** > **Content** > **Certificates** to export the certificate) and save it by the name of **obs.cer**.

Step 2 Run the **openssl x509 -inform der -in obs.cer -out obs.pem** command to convert the CER file to a PEM file.

Step 3 Modify the configuration code of the instance of ObsClient and enable server-side verification. Sample code is as follows:

```
# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint',# Set the protocol to be HTTPS.
    ssl_verify= 'your_cafile_dir/obs.pem' # Configure the path to the .pem root certificate.
)
```

----End

5 Fault Locating

5.1 Methods

If problems occur when using the OBS Python SDK, you can perform the following steps to analyze and locate the problems.

Step 1 Make sure that the OBS Python SDK is the latest version. Click **here** to download the latest version.

Step 2 You are recommended to enable the logging function of the OBS Python SDK. For details about how to enable the function, see the **Log Analysis** section. The recommended log level is WARN.

Step 3 Make sure that the program code of the OBS Python SDK complies with **General Examples of ObsClient**. All ObsClient APIs are processed with exception handling and offer HTTP status codes for the returned results. The following is an example code of uploading an object:

```
# Import the module.
from obs import ObsClient

# ObsClient is thread-safe and can be simultaneously used by multiple threads.
# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
try:
    resp = obsClient.putContent('bucketname', 'objectname', 'Hello OBS')
    # If no exception occurs, the API call is complete. Check the HTTP status code using the SDK common result objects.
    if resp.status < 300:
        # Optional: After the API is successfully called, record the HTTP status code and request ID returned by the server.
        print('status:', resp.status)
        print('requestId:', resp.requestId)
    else:
        # Recommended: If the call fails, record the HTTP status code, server-side error code, and request ID returned by the server.
        print('status:', resp.status)
        print('errorCode:', resp.errorCode)
        print('requestId:', resp.requestId)
 except Exception as e:
    import traceback
```



```
    # Recommended: When an exception occurs, record the stack information. The common scenario is
network exception.
    print(traceback.format_exc())
```

NOTE

You can view the details about the SDK common result objects **here**.

Step 4 If an exception occurs during the ObsClient API call, check the network between the client and the OBS server. If the network is normal, collect the stack information of the exception and contact the OBS client O&M team to locate the cause of the exception.

Step 5 If an ObsClient API call fails, obtain the **HTTP status code** and **OBS server-side error code** from the SDK common result objects, and compare them to locate the cause.

Step 6 If the cause cannot be found in step 5, obtain the request ID returned by the OBS server from the SDK common result objects and contact the OBS server O&M team to locate the cause.

Step 7 If the request ID cannot be obtained from the SDK common result objects, contact the OBS client O&M team to locate the failure cause.

----End

5.2 Notable Issues

SignatureDoesNotMatch

```
status: 403
errorCode: SignatureDoesNotMatch
```

Possible causes are as follows:

1. The SK input into ObsClient initialization is incorrect. Solution: Make sure that the SK is correct.
2. This problem is caused by a bug in the OBS Python SDK of an earlier version. Solution: Upgrade the SDK to the latest version.

MethodNotAllowed

```
status: 405
errorCode: MethodNotAllowed
```

This error occurs because a feature on which the ObsClient API depends has not been rolled out on the requested OBS server. Contact the OBS O&M team for further confirmation.

BucketAlreadyOwnedByYou

```
status: 409
errorCode: BucketAlreadyOwnedByYou
```

In OBS, a bucket name must be globally unique. Solution: If **ObsClient.createBucket** fails to be called, check whether the bucket exists. You can use either of the following methods to check whether a bucket exists:

Method 1 (recommended): Call **ObsClient.listBuckets** to query the list of all buckets that you own and check whether the bucket exists.

Method 2: Call **ObsClient.getBucketMetadata**, check whether the HTTP status code is 404, and determine the existence of the bucket. If the HTTP status code is 404, the bucket does not exist. If the HTTP status code is not 404, refer to **HTTP Status Codes** for more information.

NOTE

ObsClient.getBucketMetadata can query only buckets in the current region, while **ObsClient.listBuckets** can query buckets in all regions.

BucketAlreadyExists

```
status: 409
errorCode: BucketAlreadyExists
```

In OBS, a bucket name must be globally unique. Solution: Create a bucket with another name.

Connection Timeout

If **TimeoutError** occurs or the error information **Errno 10060** is returned to the ObsClient API call, the connection times out. Possible causes are as follows:

1. The endpoint input into ObsClient initialization is incorrect. Solution: Make sure that the endpoint is correct.
2. The network between the OBS client and OBS server is abnormal. Solution: Check the health status of the network.
3. The OBS domain name resolved by DNS is inaccessible. Solution: Contact the OBS O&M team.

Failed to Obtain the Error Code from the SDK Common Result Objects

This error occurs usually because the call of **ObsClient.getBucketMetadata** or **ObsClient.getObjectMetadata** fails. In this scenario, the server does not return an error code because the request method used in the background is HEAD. Solution: Obtain the HTTP status code from the SDK common result objects to analyze the possible cause. For example, 403 indicates that the user does not have the access permission, and 404 indicates that the bucket or object does not exist. If the cause cannot be located, obtain the request ID returned by the OBS server from the SDK common result objects and contact the OBS O&M team.

SDK Log Initialization Failed

```
LOGCONF is not in secs:[]
```

Possible causes are as follows:

1. The path of the log configuration file passed by **ObsClient.initLog** is incorrect. Solution: Make sure that the path of the log configuration file is correct.
2. The user running the Python process does not have the permission to read the log configuration file. Solution: Assign the user the permission to read the log configuration file.

Missing Module for the ObsClient Initialization

```
ImportError: No module named obs
```

This error occurs if the OBS Python SDK is incorrectly installed. Solution: Make sure that the SDK is installed correctly. See **Installing the SDK**.

6 Bucket Management

6.1 Creating a Bucket

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

Creating a Bucket Directly

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
# Create a bucket.
resp = obsClient.createBucket('bucketname')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


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 (cn-north-1).

NOTICE

- When creating a bucket, you do not need to specify the region if the endpoint belongs to the default region (cn-north-1). If the endpoint belongs to a region other than the default one, you need to specify the region to which the endpoint belongs. For more information, refer to **Creating a Bucket with Parameters Specified**. 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 region for the bucket. OBS provides three storage classes for buckets. For details, see **6.11 Setting or Obtaining the Storage Class of a Bucket**. Sample code is as follows:

```
# Import the module.
from obs import ObsClient
from obs import StorageClass
from obs import HeadPermission

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Import the additional header module for creating buckets.
from obs import CreateBucketHeader
header = CreateBucketHeader()
# Set the ACL to public-read. The default ACL is private.
header.aclControl = HeadPermission.PUBLIC_READ
# Set the storage class to OBS Archive.
header.storageClass = StorageClass.COLD
# Set the bucket location.
location = 'bucketlocation'

# Create a bucket.
resp = obsClient.createBucket('bucketname', header=header, location=location)
```



```
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

6.2 Listing Buckets

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# List buckets.
resp = obsClient.listBuckets(isQueryLocation=True)
if resp.status < 300:
    print('requestId:', resp.requestId)
    index = 1
    for bucket in resp.body.buckets:
        print('bucket [' + str(index) + ']')
        print('name:', bucket.name)
        print('create_date:', bucket.create_date)
        print('location:', bucket.location)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

- Obtained bucket names are listed in the lexicographical order.
- Set **isQueryLocation** to **True** and then you can query the bucket location when listing buckets.

6.3 Deleting a Bucket

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Delete a bucket.
resp = obsClient.deleteBucket('bucketname')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


NOTE

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

6.4 Identifying Whether a Bucket Exists

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.headBucket('bucketname')
if resp.status < 300:
    # The bucket exists.
    print('Bucket exists')
elif resp.status == 404:
    # The bucket does not exist.
    print('Bucket does not exist')
```

NOTE

If the returned HTTP status code is **404**, the bucket does not exist.

6.5 Obtaining Bucket Metadata

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
# Obtain the bucket metadata.
resp = obsClient.getBucketMetadata('bucketname', 'http://www.a.com', 'x-obs-header')
if resp.status < 300:
    print('requestId:', resp.requestId)
    print('storageClass:', resp.body.storageClass)
    print('accessContorlAllowOrigin:', resp.body.accessContorlAllowOrigin)
    print('accessContorlMaxAge:', resp.body.accessContorlMaxAge)
    print('accessContorlExposeHeaders:', resp.body.accessContorlExposeHeaders)
    print('accessContorlAllowMethods:', resp.body.accessContorlAllowMethods)
    print('accessContorlAllowHeaders:', resp.body.accessContorlAllowHeaders)
else:
    print('status:', resp.status)
```


6.6 Managing Bucket ACLs

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 the OBS Python SDK
READ	A grantee with this permission for a bucket can obtain the list of objects in the bucket and the metadata of the bucket. A grantee with this permission for an object can obtain the object content and metadata.	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.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.READ_ACP
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.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.FULL_CONTROL

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

Permission	Description	Value in the OBS Python SDK
private	Indicates that 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.	HeadPermission.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.	HeadPermission.PUBLIC_READ
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.	HeadPermission.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.	HeadPermission.PUBLIC_READ_DELIVERED

Permission	Description	Value in the OBS Python SDK
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.	HeadPermission.PUBLIC_READ_WRITE_DELIVERED

Specifying a Pre-defined Access Control Policy During Bucket Creation

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import CreateBucketHeader
from obs import HeadPermission
# Set the bucket ACL to public-read-write.
header = CreateBucketHeader(aclControl=HeadPermission.PUBLIC_READ_WRITE)
# Create a bucket.
resp = obsClient.createBucket('bucketname', header=header)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Setting a Pre-defined Access Control Policy for a Bucket

Sample code:

```
# Import the module.
from obs import ObsClient
from obs import HeadPermission

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Set the bucket ACL to private.
resp = obsClient.setBucketAcl('bucketname', aclControl=HeadPermission.PRIVATE)
```



```
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Directly Setting a Bucket ACL

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import ACL
from obs import Owner
from obs import Grant, Permission
from obs import Grantee, Group

owner = Owner(owner_id='ownerid')

grantee1 = Grantee(grantee_id='userid')
grantee2 = Grantee(group=Group.ALL_USERS)
# Grant READ and WRITE permissions to a specified user.
grant1 = Grant(grantee=grantee1, permission=Permission.READ)
grant2 = Grant(grantee=grantee1, permission=Permission.WRITE)
# Grant the READ and READ_ACP permissions to all users.
grant3 = Grant(grantee=grantee2, permission=Permission.READ)
grant4 = Grant(grantee=grantee2, permission=Permission.READ_ACP)

acl = ACL(owner=owner, grants=[grant1, grant2, grant3, grant4])
# Directly set the bucket ACL.
resp = obsClient.setBucketAcl('bucketname', acl)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

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

Obtaining a Bucket ACL

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.getBucketAcl('bucketname')
```



```
if resp.status < 300:
    print('requestId:', resp.requestId)
    print('owner_id:', resp.body.owner.owner_id)
    print('owner_name:', resp.body.owner.owner_name)
    index = 1;
    for grant in resp.body.grants:
        print('grant [' + str(index) + ']')
        print('grant_id:', grant.grantee.grantee_id)
        print('grant_name:', grant.grantee.grantee_name)
        print('group:', grant.grantee.group)
        print('permission:', grant.permission)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

6.7 Management Bucket Policies

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 Overview**.

Setting a Bucket Policy

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.setBucketPolicy('bucketname', 'your policy')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
```



```
)

resp = obsClient.getBucketPolicy('bucketname')
if resp.status < 300:
    print('requestId:', resp.requestId)
    print('policy:', resp.body)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Deleting a Bucket Policy

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.deleteBucketPolicy('bucketname')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

6.8 Obtaining a Bucket Location

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.getBucketLocation('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
    print('location:', resp.body.location)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

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

6.9 Obtaining Storage Information About a Bucket

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.getBucketStorageInfo('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
    print('size:', resp.body.size)
    print('objectNumber:', resp.body.objectNumber)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

6.10 Setting or Obtaining a Bucket Quota

Setting a Bucket Quota

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
# Set the bucket quota to 100 MB.
resp = obsClient.setBucketQuota('bucketname', 100 * 1024 * 1024)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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 the bucket quota. Sample code is as follows:


```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.getBucketQuota('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
    print('quota:', resp.body.quota)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

6.11 Setting or Obtaining the Storage Class of a Bucket

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 Python 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.	StorageClass.STANDARD
OBS Infrequent Access	Is applicable to storing semi-frequently accessed (less than 12 times a year) data requiring quick response.	StorageClass.WARM
OBS Archive	Is applicable to archiving rarely-accessed (once a year) data.	StorageClass.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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
```



```
# Set the storage class to OBS Infrequent Access.
resp = obsClient.setBucketStoragePolicy('bucketname', StorageClass.WARM)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.getBucketStoragePolicy('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
    print('storageClass:', resp.body.storageClass)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


7 Object Upload

7.1 Object Upload Overview

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

- **7.2 Performing a Text-Based Upload**
- **7.3 Performing a Streaming Upload**
- **7.4 Performing a File-Based Upload**
- **7.8 Performing a Multipart Upload**
- **7.10 Performing an Appendable Upload**
- **7.12 Performing a Resumable Upload**
- **7.13 Performing a Browser-Based Upload**

The SDK supports the upload of objects whose size ranges from 0 KB to 5 GB. For steaming upload, appendable upload and file-based upload, data to be uploaded at a time 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 the bucket, its URL does not contain directory levels.

7.2 Performing a Text-Based Upload

Text-based upload is used to directly upload character strings. You can call **ObsClient.putContent** to upload character strings to OBS. Sample code is as follows:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
```



```
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.putContent('bucketname', 'objectname', content='Hello OBS')

if resp.status < 300:
    print('requestId:', resp.requestId)
    print('etag:', resp.body.etag)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

7.3 Performing a Streaming Upload

Streaming upload uses readable objects that contain the **read** property as data sources. Sample code is as follows:

Uploading a Network Stream

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

import sys
if sys.version_info.major == 2 or not sys.version > '3':
    import httplib
else:
    import http.client as httplib

conn = httplib.HTTPConnection('www.a.com', 80)
conn.request('GET', '/')
content = conn.getresponse()

resp = obsClient.putContent('bucketname', 'objectname', content=content)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Uploading a File Stream

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
content = open('localfile', 'rb')

resp = obsClient.putContent('bucketname', 'objectname', content=content)

if resp.status < 300:
    print('requestId:', resp.requestId)
```



```
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTICE

- When this upload mode is used, the value of **content** must be a readable object that contains the **read** property.
- When uploading file streams, you must open files in **rb** or **rb+** mode.
- To upload a large file, you are advised to use **multipart upload**.
- The content to be uploaded cannot exceed 5 GB.

7.4 Performing a File-Based Upload

File-based upload uses local files as data sources. You can call **ObsClient.putFile** to upload a local file. Sample code is as follows:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.putFile('bucketname', 'objectname', file_path='localfile') # localfile indicates the local path of the file to be uploaded. You need to specify the file name.

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

The content to be uploaded cannot exceed 5 GB.

7.5 Obtaining Upload Progress

You can set the callback function to obtain upload progress. Sample code is as follows:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

def callback(transferredAmount, totalAmount, totalSeconds):
    # Obtain the average upload rate (KB/s).
    print(transferredAmount * 1.0 / totalSeconds / 1024)
```



```
    # Obtain the upload progress in percentage.
    print(transferredAmount * 100.0 / totalAmount)

resp = obsClient.putFile('bucketname', 'objectname', file_path='localfile', progressCallback=callback)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

You can query the upload progress when uploading an object in streaming, file-based, multipart, appendable, or resumable mode.

7.6 Creating a Folder

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.

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.putContent('bucketname', 'parent_directory/', content=None)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)

# Create an object in the folder.
resp = obsClient.putContent('bucketname', 'parent_directory/objectname', content='Hello OBS')

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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.

7.7 Setting Object Properties

You can set properties for an object when uploading it. Object properties include the object length, MIME type, MD5 value (for verification), storage class, and

customized metadata. You can set properties for an object that is being uploaded in text-based, 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.	binary/octet-stream
Content-MD5	Indicates the base64-encoded digest of the object data. It is provided for the OBS server to verify data integrity.	None
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.	None
Customized metadata	Indicates the user-defined description of the object. It is used to facilitate the customized management on the object.	None

Setting the Length for an Object

You can call **PutObjectHeader.contentLength** to set the object length. Sample code is as follows:

```
# Import the module.
from obs import ObsClient
```



```
# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import PutObjectHeader
headers = PutObjectHeader()
# Only 100 bytes of the object will be uploaded.
headers.contentLength = 100
content = open('localfile', 'rb')
resp = obsClient.putContent('bucketname', 'objectname', content=content, headers=headers)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Setting the MIME Type for an Object

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import PutObjectHeader
headers = PutObjectHeader()
# Upload an image.
headers.contentType = 'image/jpeg'
resp = obsClient.putFile('bucketname', 'objectname.jpg', file_path='localimage.jpg', headers=headers)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

If this property is not specified, SDK will automatically identify the MIME type according to the suffix of the uploaded object. For example, if the suffix of the 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 **PutObjectHeader.md5** to set the MD5 value for an object. Sample code is as follows:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
```



```
)

from obs import PutObjectHeader
headers = PutObjectHeader()
# Set the MD5 value.
headers.md5 = 'your md5 which should be encoded by base64'
resp = obsClient.putFile('bucketname', 'objectname.jpg', file_path='localimage.jpg', headers=headers)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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 and an HTTP 400 error will be returned.
- If the MD5 value is not specified, the OBS server will skip MD5 value verification.

Setting the Storage Class for an Object

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import PutObjectHeader
from obs import StorageClass
headers = PutObjectHeader()
# Set the storage class to OBS Archive.
headers.storageClass = StorageClass.COLD
resp = obsClient.putFile('bucketname', 'objectname', file_path='localfile', headers=headers)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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 use the **metadata** parameter to customize the metadata for the object. Sample code is as follows:


```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
metadata = {'property1' : 'property-value1', 'property2' : 'property-value2'}
resp = obsClient.putFile('bucketname', 'objectname', file_path='localfile', metadata=metadata)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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. The total metadata size cannot exceed 8 KB.
- The customized object metadata can be obtained by using **ObsClient.getObjectMetadata**. For details, see **Obtaining Object Metadata**.
- When you call **ObsClient.getObject** to download an object, its customized metadata will also be downloaded.

7.8 Performing a Multipart Upload

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.

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

metadata = {'property1' : 'property-value1', 'property2' : 'property-value2'}
resp = obsClient.initiateMultipartUpload('bucketname', 'objectname', contentType='text/plain',
metadata=metadata)

if resp.status < 300:
    print('requestId:', resp.requestId)
    print('uploadId:', resp.body.uploadId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

- When initializing a multipart upload, you can use the **contentType** and **metadata** parameters to respectively set the MIME type and customize the metadata of an object, besides the object name and owning bucket.
- After the API for initializing a multipart upload is called, the upload ID will be returned. This ID 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 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.

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
uploadId = 'upload id from initiateMultipartUpload'
// File to be uploaded. localfile is the path of the local file to be uploaded. You need to specify the file
name.
file_path = 'localfile'

# Part number of the first part
partNumber = 1
# Offset of the first part
offset = 0
# Part size of the first part
```



```
partSize = 5 * 1024 * 1024
# Upload the first part.
resp = obsClient.uploadPart('bucketname', 'objectname', partNumber=partNumber, uploadId=uploadId,
                            offset=offset, partSize=partSize, object=file_path, isFile=True)

if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain the ETag after the upload.
    print('etag:', resp.body.etag)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)

# Part number of the second part
partNumber = 2
# Offset of the second part
offset = 5 * 1024 * 1024
# Part size of the second part
partSize = 5 * 1024 * 1024
# Upload the second part.
resp = obsClient.uploadPart('bucketname', 'objectname', partNumber=partNumber, uploadId=uploadId,
                            offset=offset, partSize=partSize, object=file_path, isFile=True, isAttachMd5=True)

if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain the ETag after the upload.
    print('etag:', resp.body.etag)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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 **isAttachMd5** to **True** (the default value is **False**) to make SDK to automatically calculate the MD5 value 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 SDK to verify the data integrity.
- You can use the **md5** parameter to set the MD5 value of the uploaded data directly. If this parameter is set, the **isAttachMd5** parameter becomes ineffective.
- Part numbers range from 1 to 10000. If the part number you set is out of this range, OBS will return error **400 Bad Request**.
- The minimum part size supported by an OBS 3.0 bucket is 100 KB, and the minimum part size supported by an OBS 2.0 bucket is 5 MB. You are advised to perform multipart upload to OBS 3.0 buckets.

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.

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
```



```
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import CompleteMultipartUploadRequest, CompletePart

part1 = CompletePart(partNum=1, etag='etag1')
part2 = CompletePart(partNum=2, etag='etag2')

completeMultipartUploadRequest = CompleteMultipartUploadRequest()
completeMultipartUploadRequest.parts = [part1, part2]

resp = obsClient.completeMultipartUpload('bucketname', 'objectname', 'uploadid',
completeMultipartUploadRequest)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

- In the preceding code, **CompleteMultipartUploadRequest.parts** 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 in a multipart upload:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

import platform, os, threading, multiprocessing
IS_WINDOWS = platform.system() == 'Windows' or os.name == 'nt'

bucketName = 'bucketname'
objectKey = 'objectname'
filePath = 'localfile'

def doUploadPart(partETags, bucketName, objectKey, partNumber, uploadId, filePath, partSize, offset):
    resp = obsClient.uploadPart(bucketName, objectKey, partNumber, uploadId, filePath, isFile=True,
partSize=partSize, offset=offset)
    if resp.status < 300:
        partETags[partNumber] = resp.body.etag
        print('\tPart#' + str(partNumber) + ' done\n')
    else:
        print('\tPart#' + str(partNumber) + ' failed\n')

if __name__ == '__main__':
    # Initialize a multipart upload.
    resp = obsClient.initiateMultipartUpload(bucketName, objectKey)
    if resp.status >= 300:
      raise Exception('initiateMultipartUpload failed')
```



```
    uploadId = resp.body.uploadId
    print(uploadId)

    # Set the part size to 100 MB.
    partSize = 100 * 1024 * 1024

    fileLength = os.path.getsize(filePath)
    # Calculate the number of parts to be uploaded.
    partCount = int(fileLength / partSize) if (fileLength % partSize == 0) else int(fileLength / partSize) + 1

    proc = threading.Thread if IS_WINDOWS else multiprocessing.Process
    partETags = dict() if IS_WINDOWS else multiprocessing.Manager().dict()
    processes = []

    #  Start uploading parts concurrently.
    for i in range(partCount):
        # Start point of the part in the file
        offset = i * partSize
        # Part size
        currPartSize = (fileLength - offset) if i + 1 == partCount else partSize
        # Part number
        partNumber = i + 1
        p = proc(target=doUploadPart, args=(partETags, bucketName, objectKey, partNumber, uploadId,
filePath, currPartSize, offset))
        p.daemon = True
        processes.append(p)

    for p in processes:
        p.start()

    # Wait until the upload is complete.
    for p in processes:
        p.join()

    if len(partETags) != partCount:
        raise Exception('Upload multiparts fail due to some parts are not finished yet')

    # Combine parts.
    from obs import CompletePart, CompleteMultipartUploadRequest
    partETags = sorted(partETags.items(), key=lambda d : d[0])
    parts = []
    for key, value in partETags:
        parts.append(CompletePart(partNum=key, etag=value))

    resp = obsClient.completeMultipartUpload(bucketName, objectKey, uploadId,
CompleteMultipartUploadRequest(parts))
    if resp.status < 300:
        print('requestId:', resp.requestId)
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        raise Exception('completeMultipartUpload failed')
```

NOTE

When uploading a large file in multipart mode, you need to use the **offset** and **partSize** parameters to set the start and end positions of each part in the file.

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.

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

obsClient.abortMultipartUpload('bucketname', 'objectname', 'upload id from initiateMultipartUpload')

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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
uploadId	Upload ID, which globally identifies a multipart upload. The value is in the returned result of **initiateMultipartUpload**.
maxParts	Maximum number of parts that can be listed per page
partNumberMarker	Part number after which listing uploaded parts begins. Only parts whose part numbers are larger than this value will be listed.

- Listing parts in simple mode

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# List uploaded parts. uploadId is obtained from initiateMultipartUpload.
obsClient.listParts('bucketname', 'objectname', uploadId='upload id from initiateMultipartUpload')

if resp.status < 300:
    print('requestId:', resp.requestId)
    index = 1
    for part in resp.body.parts:
        print('part [' + str(index) + ']')
        # Part number, specified during the upload
        print('partNumber:', part.partNumber)
        # Time when the part was last uploaded
```



```
        print('lastModified:', part.lastModified)
        # ETag of the part
        print('etag:', part.etag)
        # Part size
        print('size:', part.size)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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 **body.isTruncated** is **True** in the returned result, not all parts are listed. In such cases, you can use **body.nextPartNumberMarker** 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.

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

index = 1
nextPartNumberMarker = None
while True:
    # List uploaded parts. uploadId is obtained from initiateMultipartUpload.
    resp = obsClient.listParts('bucketname', 'objectname', uploadId='upload id from initiateMultipartUpload',
partNumberMarker=nextPartNumberMarker)
    if resp.status < 300:
        print('requestId:', resp.requestId)
        for part in resp.body.parts:
            print('part [' + str(index) + ']')
        # Part number, specified during the upload
            print('partNumber:', part.partNumber)
        # Time when the part was last uploaded
            print('lastModified:', part.lastModified)
        # ETag of the part
            print('etag:', part.etag)
        # Part size
            print('size:', part.size)
            index += 1
        if not resp.body.isTruncated:
            break
        nextPartNumberMarker = resp.body.nextPartNumberMarker
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        break
```

Listing Multipart Uploads

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

Parameter	Description
ListMultipartUploadsRequest.prefix	Prefix that the object names in the multipart uploads to be listed must contain
ListMultipartUploadsRequest.max_uploads	Maximum number of listed multipart uploads. The value ranges from 1 to 1000. If the value is not in this range, 1000 multipart uploads are listed by default.
ListMultipartUploadsRequest.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.key_marker	Object name to start with when listing multipart uploads
ListMultipartUploadsRequest.upload_id_marker	Upload ID after which the multipart upload listing begins. It is effective only when used with **key_marker** so that multipart uploads after **upload_id_marker** of **key_marker** will be listed.

- Listing multipart uploads in simple mode

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.listMultipartUploads('bucketname')
if resp.status < 300:
    print('requestId:', resp.requestId)
    index = 1
    for upload in resp.body.upload:
        print('upload [' + str(index) + ']')
        print('key:', upload.key)
        print('uploadId:', upload.uploadId)
        print('initiated:', upload.initiated)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

- Information about a maximum of 1000 multipart uploads can be listed each time. If a bucket contains more than 1000 multipart uploads and **body.isTruncated** is **True** in the returned result, not all uploads are returned. In such cases, you can use **body.nextKeyMarker** and **body.nextUploadIdMarker** 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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import ListMultipartUploadsRequest
multipart = ListMultipartUploadsRequest()

while True:
    resp = obsClient.listMultipartUploads('bucketname', multipart=multipart)
    if resp.status < 300:
        print('requestId:', resp.requestId)
        index = 1
        for upload in resp.body.upload:
            print('upload [' + str(index) + ']')
            print('key:', upload.key)
            print('uploadId:', upload.uploadId)
            print('initiated:', upload.initiated)
            index += 1

        if not resp.body.isTruncated:
            break
        multipart.key_marker = resp.body.nextKeyMarker
        multipart.upload_id_marker = resp.body.nextUploadIdMarker
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        break
```

7.9 Configuring Lifecycle Management

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import PutObjectHeader
headers = PutObjectHeader()
# When uploading an object, set the object to expire 30 days after creation.
headers.expires = 30
resp = obsClient.putContent('bucketname', 'objectname', content='Hello OBS', headers=headers)


if resp.status < 300:
    print('requestId:', resp.requestId)
    print('etag:', resp.body.etag)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)

# When initializing a multipart upload, set the object to expire 60 days after combination.
resp = obsClient.initiateMultipartUpload('bucketname', 'objectname', expires=60)
if resp.status < 300:
```



```
    print('requestId:', resp.requestId)
    print('uploadId:', resp.body.uploadId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

- The previous 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 the preceding method takes precedence over the bucket lifecycle rule.

7.10 Performing an Appendable Upload

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import AppendObjectContent
content = AppendObjectContent()
content.content = 'Hello OBS'
# Upload an object in appendable mode.
resp = obsClient.appendObject('bucketname', 'objectname', content=content)

if resp.status < 300:
    print('requestId:', resp.requestId)
    print('nextPosition:', resp.body.nextPosition)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)


content.position = resp.body.nextPosition
content.content = 'Hello OBS Again'
# Append data to the object.
resp = obsClient.appendObject('bucketname', 'objectname', content=content)

if resp.status < 300:
    print('requestId:', resp.requestId)
    print('nextPosition:', resp.body.nextPosition)
    print('etag:', resp.body.etag)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)

# Use the API for obtaining object properties to get the start position for next appending.
resp = obsClient.getObjectMetadata('bucketname', 'objectname')
if resp.status < 300:
    print('requestId:', resp.requestId)
    print('nextPosition:', resp.body.nextPosition)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


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 create an appendable object, an exception will be thrown (status code **409**) if a normal object with the same name has existed.
- 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 complete successfully, you can use **body.nextPosition** obtained from the returned result or call **ObsClient.getObjectMetadata**, to get the location for next appending.

7.11 Performing a Multipart Copy

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:

```
# Import the module.
from obs import ObsClient

destBucketName = 'bucketname'
destObjectKey = 'objectname'
sourceBucketName = 'sourcebucketname'
sourceObjectKey = 'sourceobjectname'

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

import platform, os, threading, multiprocessing
IS_WINDOWS = platform.system() == 'Windows' or os.name == 'nt'

def doCopyPart(partETags, bucketName, objectKey, partNumber, uploadId, copySource, copySourceRange):
    resp = obsClient.copyPart(bucketName=bucketName, objectKey=objectKey, partNumber=partNumber,
uploadId=uploadId, copySource=copySource, copySourceRange=copySourceRange)
    if resp.status < 300:
        partETags[partNumber] = resp.body.etag
        print('\tPart#' + str(partNumber) + ' done\n')
    else:
        print('\tPart#' + str(partNumber) + ' failed\n')

if __name__ == '__main__':
    # Initialize a multipart upload.
    resp = obsClient.initiateMultipartUpload(destBucketName, destObjectKey)
    if resp.status >= 300:
        raise Exception('initiateMultipartUpload failed')

    uploadId = resp.body.uploadId

    # Set the part size to 100 MB.
    partSize = 100 * 1024 * 1024
    # Obtain information about the large object.
    resp = obsClient.getObjectMetadata(sourceBucketName, sourceObjectKey)
    if resp.status >= 300:
        raise Exception('getObjectMetadata failed')
```



```
    objectSize = resp.body.contentLength

    # Calculate the number of parts to be copied.
    partCount = int(objectSize / partSize) if (objectSize % partSize == 0) else int(objectSize / partSize) + 1

    proc = threading.Thread if IS_WINDOWS else multiprocessing.Process

    partETags = dict() if IS_WINDOWS else multiprocessing.Manager().dict()

    processes = []

    #  Start copying parts concurrently.
    for i in range(partCount):
        # Start position for copying parts
        rangeStart = i * partSize
        # End position for copying parts
        rangeEnd = objectSize - 1 if (i + 1 == partCount) else rangeStart + partSize - 1
        # Part number
        partNumber = i + 1
        p = proc(target=doCopyPart, args=(partETags, destBucketName, destObjectKey, partNumber, uploadId,
sourceBucketName + '/' + sourceObjectKey,    str(rangeStart) + '-' + str(rangeEnd)))
        p.daemon = True
        processes.append(p)

    for p in processes:
        p.start()

    # Wait until the copy is complete.
    for p in processes:
        p.join()

    if len(partETags) != partCount:
        raise Exception('copyParts fail due to some parts are not finished yet')

    # Combine parts.
    from obs import CompletePart, CompleteMultipartUploadRequest
    partETags = sorted(partETags.items(), key=lambda d : d[0])
    parts = []
    for key, value in partETags:
        parts.append(CompletePart(partNum=key, etag=value))

    resp = obsClient.completeMultipartUpload(destBucketName, destObjectKey, uploadId,
CompleteMultipartUploadRequest(parts))
    if resp.status < 300:
        print('requestId:', resp.requestId)
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        raise Exception('completeMultipartUpload failed')
```

7.12 Performing a Resumable Upload

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 is 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
bucketName	(Mandatory) Bucket name
objectKey	(Mandatory) Object name
uploadFile	(Mandatory) Local file to be uploaded
partSize	Part size, in bytes. The value ranges from 100 KB to 5 GB and defaults to 9 MB.
taskNum	Maximum number of threads that can be concurrently executed for upload. The default value is **1**.
enableCheckpoint	Whether to enable the resumable upload mode. The default value is **False**, which indicates that this mode is disabled.
checkpointFile	File used to record the upload progress. This parameter is effective only in the resumable upload mode. If the value is **None**, the file is in the same directory as the local file to be uploaded.
metadata	Object metadata
checkSum	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 of the file will not be verified.
progressCallback	Callback function for obtaining the upload progress

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Set the local file to be uploaded. You must specify the file name.
uploadFile = 'localfile'
# Set the maximum number of threads that can be concurrently executed for upload.
taskNum = 5
# Set the part size to 10 MB.
partSize = 10 * 1024 * 1024
# Enable resumable upload.
enableCheckpoint = True
# Perform a resumable upload.
```



```
try:
    resp = obsClient.uploadFile('bucketname', 'objectname', uploadFile, partSize, taskNum, enableCheckpoint)
    if resp.status < 300:
        print('requestId:', resp.requestId)
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        # When the upload fails, you can call the API for resumable upload to continue to upload the file.
except:
    # When an exception occurs, you can call the API for resumable upload to continue to upload the file.
    pass
```

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, the API for resumable upload degrades to the simple encapsulation of multipart upload, and no checkpoint file will be generated.
- **checkpointFile** and **checkSum** are effective only when **enableCheckpoint** is **True**.

7.13 Performing a Browser-Based Upload

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 **post_object_sample**. You can also perform a browser-based upload as follows:

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.

```
# Import the module.
from obs import ObsClient
```



```
# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='http://your-endpoint'
)

# Set parameters for the form.
formParams = {}
# Set the object ACL to public-read.
formParams['x-obs-acl'] = 'public-read'
# Set the MIME type for the object.
formParams['content-type'] = 'text/plain'

# Set the validity period for the browser-based upload request, in seconds.
resp = obsClient.createPostSignature(expires=3600, formParams=formParams)

# Obtain the request parameters.
print('\t', resp['policy'])
print('\t', resp['signature'])
```

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
<!-- ObjectACL -->
<input type="text" name="x-obs-acl" value="public-read" />
<p>
Content-Type
<!-- MIME type of the object -->
<input type="text" name="content-type" value="text/plain" />
<p>
<!-- Base64 code of policy -->
<input type="hidden" name="policy" value="*** Provide your policy ***" />
<!-- AK -->
<input type="hidden" name="AccessKeyId" value="*** Provide your access key ***"/>
<!-- Signature string -->
<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**.

8 Object Download

8.1 Object Download Overview

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

- **8.2 Performing a Binary Download**
- **8.3 Performing a Streaming Download**
- **8.4 Performing a File-Based Download**
- **8.5 Performing a Partial Download**
- **8.11 Performing a Resumable Download**

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

8.2 Performing a Binary Download

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.getObject('bucketname', 'objectname', loadStreamInMemory=True)
if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain the object content.
    print('buffer:', resp.body.buffer)
    print('size:',resp.body.size)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


NOTE

In a binary download, if **loadStreamInMemory** is set to **True**, the object content is contained in the **body.buffer** field in returned result.

8.3 Performing a Streaming Download

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.getObject('bucketname', 'objectname', loadStreamInMemory=False)
if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain the object content.
    while True:
        chunk = resp.body.response.read(65536)
        if not chunk:
            break
        print(chunk)
    resp.body.response.close()
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

If you perform a stream download in the condition that **loadStreamInMemory** is set to **False**, the **body.response** field in the returned result is a readable stream. You can read the object content and save the content to a local file or the memory through the stream.

NOTICE

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

8.4 Performing a File-Based Download

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.getObject('bucketname', 'objectname', downloadPath='localfile')
if resp.status < 300:
```



```
    print('requestId:', resp.requestId)
    print('url:', resp.body.url)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

- Use the **downloadPath** parameter to set the save path for the file to be downloaded.
- When **loadStreamInMemory** is set to **True**, **downloadPath** is ineffective.

NOTICE

If the value of **downloadPath** is a file path instead of a folder path, the object is downloaded as the file.

8.5 Performing a Partial Download

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import GetObjectHeader
headers = GetObjectHeader()
# Set the start point and end point.
headers.range = '0-1000'

resp = obsClient.getObject('bucketname', 'objectname', loadStreamInMemory=True, headers=headers)
if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain the object content.
    print('buffer:', resp.body.buffer)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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 **concurrent_download_object_sample**.

8.6 Obtaining Download Progress

You can set the callback function to obtain download progress. Sample code is as follows:


```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
def callback(transferredAmount, totalAmount, totalSeconds):
    # Obtain the average upload rate (KB/s).
    print(transferredAmount * 1.0 / totalSeconds / 1024)
    # Obtain the download progress in percentage.
    print(transferredAmount * 100.0 / totalAmount)

resp = obsClient.getObject('bucketname', 'objectname', progressCallback=callback)
if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain the object content.
    while True:
        chunk = resp.body.response.read(65536)
        if not chunk:
            break
        print(chunk)
    resp.body.response.close()
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

You can obtain the download progress when downloading an object in binary, streaming, file-based, or resumable mode.

8.7 Performing a Conditioned Download

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

You can set the following conditions:

Parameter	Description	Format
GetObjectHeader.if_modified_since	Returns the object if it is modified after the time specified by this parameter; otherwise, an error code is returned.	This parameter must conform to the HTTP time format specified in http://www.ietf.org/rfc/rfc2616.txt.
GetObjectHeader.if_unmodified_since	Returns the object if it remains unchanged since the time specified by this parameter; otherwise, an error code is returned.	This parameter must conform to the HTTP time format specified in http://www.ietf.org/rfc/rfc2616.txt.

Parameter	Description	Format
GetObjectHeader.if_match	Returns the source object if its ETag is the same as the one specified by this parameter; otherwise, an error code is returned.	Character string
GetObjectHeader.if_none_match	Returns the source object if its ETag is different from the one specified by this parameter; otherwise, an error code is returned.	Character string

NOTE

- The ETag of the source object is the MD5 check value of the source object content.
- If a request includes **if_unmodified_since** or **if_match** and the specified condition is not met, the object download will fail with error code **412 Precondition Failed**.
- If a request includes **if_modified_since** or **if_none_match** and the specified condition is not met, the object download will fail with error code **304 Not Modified** returned.

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import GetObjectHeader
headers = GetObjectHeader()
headers.if_modified_since = 'Wed, 04 Jul 2018 08:54:53 GMT'

resp = obsClient.getObject('bucketname', 'objectname', loadStreamInMemory=True, headers=headers)
if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain the object content.
    print('buffer:', resp.body.buffer)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

8.8 Rewriting Response Headers

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

Parameter	Description
GetObjectRequest.content_type	Rewrites **Content-Type** in HTTP/HTTPS responses.

Parameter	Description
GetObjectRequest.content_language	Rewrites **Content-Language** in HTTP/HTTPS responses.
GetObjectRequest.expires	Rewrites **Expires** in HTTP/HTTPS responses.
GetObjectRequest.cache_control	Rewrites **Cache-Control** in HTTP/HTTPS responses.
GetObjectRequest.content_disposition	Rewrites **Content-Disposition** in HTTP/HTTPS responses.
GetObjectRequest.content_encoding	Rewrites **Content-Encoding** in HTTP/HTTPS responses.

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import GetObjectRequest

getObjectRequest = GetObjectRequest()
getObjectRequest.content_type = 'image/jpeg'

resp = obsClient.getObject('bucketname', 'objectname', loadStreamInMemory=True,
getObjectRequest=getObjectRequest)
if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain the rewritten response headers.
    print('contentType:', resp.body.contentType)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

8.9 Obtaining Customized Metadata

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Upload the object and customize the metadata.
obsClient.putContent('bucketname', 'objectname', metadata={'property' : 'property-value'})
```



```
# Download the object.
resp = obsClient.getObject('bucketname', 'objectname', loadStreamInMemory=True)
if resp.status < 300:
    # Obtain the customized metadata of the object.
    print('property:', dict(resp.header).get('property'))
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

8.10 Downloading an Archive Object

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 Python SDK
Expedited	Data can be restored within 1 to 5 minutes.	RestoreTier.EXPEDITED
Standard	Data can be restored within 3 to 5 hours. This is the default option.	RestoreTier.STANDARD

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

```
# Import the module.
from obs import ObsClient
from obs import RestoreTier

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Restore an Archive object.
resp = obsClient.restoreObject('bucketname', 'objectname', days=1, tier=RestoreTier.EXPEDITED)
if resp.status < 300:
    print('requestId:', resp.requestId)
    # Wait until the object is restored.
    import time
    time.sleep(60 * 6)
# Download an object.
    resp2 = obsClient.getObject('bucketname', 'objectname', loadStreamInMemory=True)
    if resp2.status < 300:
        print('requestId:', resp2.requestId)
        # Obtain the object content.
        print('buffer:', resp2.body.buffer)
        print('size:',resp2.body.size)
    else:
        print('errorCode:', resp2.errorCode)
        print('errorMessage:', resp2.errorMessage)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


NOTE

- The object specified in **ObsClient.restoreObject** must be in the OBS Archive storage class. Otherwise, an error will be reported when you call this API.
- Use the **days** parameter to specify the retention period of to-be-restored objects, the parameter value ranges from **1** to **30**. Use the **tier** parameter to specify the restore option that indicates the time consumed for restoring objects.

8.11 Performing a Resumable Download

Downloading large files often fails due to poor network conditions or program breakdowns. Downloading large files often fails due to poor network conditions or program breakdowns. 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 is 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
bucketName	(Mandatory) Bucket name
objectKey	(Mandatory) Object name
downloadFile	Full path of the local directory to which the object is downloaded. If the value is **None**, the downloaded object is saved in the directory where the program is executed.
partSize	Part size, in bytes. The value ranges from 100 KB to 5 GB and defaults to 5 MB.
taskNum	Maximum number of threads that can be concurrently executed for download. The default value is **1**.
enableCheckpoint	Whether to enable the resumable download mode. The default value is **False**, which indicates that this mode is disabled.
checkpointFile	File used to record the download progress. This parameter is effective only in the resumable download mode. If the value is **None**, the file is in the same local directory as the downloaded object.
versionId	Object version ID

Parameter	Description
progressCallback	Callback function for obtaining the download progress

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Set the local path to which the object is downloaded.
downloadFile = 'localfile'
# Set the maximum number of threads that can be concurrently executed for download.
taskNum = 5
# Set the part size to 10 MB.
partSize = 10 * 1024 * 1024
# Enable the resumable download mode.
enableCheckpoint = True
# Perform a resumable download.
try:
    resp = obsClient.downloadFile('bucketname', 'objectname', downloadFile, partSize, taskNum,
enableCheckpoint)
    if resp.status < 300:
        print('requestId:', resp.requestId)
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        # When the download fails, you can call the API for resumable download to continue the download.
except:
    # When an exception occurs, you can call the API for resumable download to continue the download.
    pass
```

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, the API for resumable download degrades to the simple encapsulation of partial download, and no checkpoint file will be generated.
- **checkpointFile** is effective only when **enableCheckpoint** is **True**.

8.12 Processing an Image

OBS can be used to process images in a stable, secure, efficient, easy-of-use, and cost-efficient manner. If the object to be downloaded is an image, you can pass the image processing parameters to operate it, 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:

```
# Import the module.
from obs import ObsClient
from obs import GetObjectRequest

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Resize and then rotate the image.
getObjectRequest = GetObjectRequest(imageProcess='image/resize,m_fixed,w_100,h_100/rotate,90')
resp = obsClient.getObject('bucketname', 'objectname.jpg', downloadPath='localimage.jpg',
getObjectRequest=getObjectRequest)
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

- Use **GetObjectRequest.imageProcess** 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.

9 Object Management

9.1 Obtaining Object Properties

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

```
# Import the module.
from obs import ObsClient
# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.getObjectMetadata('bucketname', 'objectname')
if resp.status < 300:
    print('requestId:', resp.requestId)
    print('contentType:', resp.body.contentType)
    print('contentLength:', resp.body.contentLength)
    print('property:', dict(resp.header).get('property'))
else:
    print('requestId:', resp.requestId)
```

9.2 Managing Object ACLs

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:

```
# Import the module.
from obs import ObsClient
```



```
# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import PutObjectHeader
from obs import HeadPermission
headers = PutObjectHeader()
# Set the object ACL to public-read.
headers.acl = HeadPermission.PUBLIC_READ

resp = obsClient.putFile('bucketname', 'objectname', 'localfile', headers=headers)
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Setting a Pre-defined Access Control Policy for an Object

Sample code:

```
# Import the module.
from obs import ObsClient
# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Set the object ACL to private.
resp = obsClient.setObjectAcl('bucketname', 'objectname', aclControl='private')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Directly Setting an Object ACL

Sample code:

```
# Import the module.
from obs import ObsClient
# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import ACL
from obs import Owner
from obs import Grant, Permission
from obs import Grantee, Group
owner = Owner(owner_id='ownerid')

# Grant all permissions to a specified user.
grant0 = Grant(grantee=Grantee(grantee_id='userid'), permission=Permission.FULL_CONTROL)

# Grant the READ permission to all users.
grant1 = Grant(grantee=Grantee(group=Group.ALL_USERS), permission=Permission.READ)

acl = ACL(owner=owner, grants=[grant0, grant1])
```



```
resp = obsClient.setObjectAcl('bucketname', 'objectname', acl=acl)
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

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

Obtaining an Object ACL

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

```
# Import the module.
from obs import ObsClient
# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.getObjectAcl('bucketname', 'objectname')
if resp.status < 300:
    print('requestId:', resp.requestId)
    print('owner_id:', resp.body.owner.owner_id)
    print('owner_name:', resp.body.owner.owner_name)
    index = 1
    for grant in resp.body.grants:
        print('grant [' + str(index) + ']')
        print('grantee_id:', grant.grantee.grantee_id)
        print('grantee_name:', grant.grantee.grantee_name)
        print('group:', grant.grantee.group)
        print('permission:', grant.permission)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

9.3 Listing Objects

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

The following table describes the parameters involved in this API.

Parameter	Description
prefix	Name prefix that the objects to be listed must contain
marker	Object name to start with when listing objects in a bucket. All objects are listed in the lexicographical order.
max_keys	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.

Parameter	Description
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.)

Listing Objects in Simple Mode

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.listObjects('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
    index = 1
    for content in resp.body.contents:
        print('object [' + str(index) + ']')
        print('key:', content.key)
        print('owner_id:', content.owner.owner_id)
        print('owner_name:', content.owner.owner_name)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

- A maximum of 1000 objects can be listed each time. If a bucket contains more than 1000 objects and **body.is_truncated** is **True** in the returned result, not all objects are listed. In such cases, you can use **body.next_marker** 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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
```



```
# Set the number of objects to be listed to 100.
resp = obsClient.listObjects('bucketname', max_keys=100)

if resp.status < 300:
    print('requestId:', resp.requestId)
    index = 1
    for content in resp.body.contents:
        print('object [' + str(index) + ']')
        print('key:', content.key)
        print('owner_id:', content.owner.owner_id)
        print('owner_name:', content.owner.owner_name)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Listing Objects by Specifying a Prefix

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Set the prefix to prefix and the number of objects to be listed to 100.
resp = obsClient.listObjects('bucketname', max_keys=100, prefix='prefix')

if resp.status < 300:
    print('requestId:', resp.requestId)
    index = 1
    for content in resp.body.contents:
        print('object [' + str(index) + ']')
        print('key:', content.key)
        print('owner_id:', content.owner.owner_id)
        print('owner_name:', content.owner.owner_name)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Listing Objects by Specifying the Start Position

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Set that 100 objects whose names are following "test" in lexicographical order will be listed.
resp = obsClient.listObjects('bucketname', max_keys=100, marker='test')

if resp.status < 300:
    print('requestId:', resp.requestId)
    index = 1
    for content in resp.body.contents:
```



```
        print('object [' + str(index) + ']')
        print('key:', content.key)
        print('owner_id:', content.owner.owner_id)
        print('owner_name:', content.owner.owner_name)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Listing All Objects in Paging Mode

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Set the number of objects displayed per page to 100.
max_keys = 100
index = 1
marker = None
while True:
    resp = obsClient.listObjects('bucketname', max_keys=max_keys, marker=marker)
    if resp.status < 300:
        print('requestId:', resp.requestId)
        for content in resp.body.contents:
            print('object [' + str(index) + ']')
            print('key:', content.key)
            print('owner_id:', content.owner.owner_id)
            print('owner_name:', content.owner.owner_name)
            index += 1
        if not resp.body.is_truncated:
            break
        marker = resp.body.next_marker
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        break
```

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Set the number of objects displayed per page to 100.
max_keys = 100
index = 1
marker = None
# Set the prefix to "dir/".
```



```
prefix = 'dir/'
while True:
    resp = obsClient.listObjects('bucketname', max_keys=max_keys, marker=marker, prefix=prefix)
    if resp.status < 300:
        print('requestId:', resp.requestId)
        for content in resp.body.contents:
            print('object [' + str(index) + ']')
            print('key:', content.key)
            print('owner_id:', content.owner.owner_id)
            print('owner_name:', content.owner.owner_name)
            index += 1
        if not resp.body.is_truncated:
            break
        marker = resp.body.next_marker
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        break
```

Listing All Objects According to Folders in a Bucket

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

def listObjectsByPrefix(resp):
    for prefix in resp.body.commonPrefixs:
        subresp = obsClient.listObjects('bucketname', delimiter='/', prefix=prefix.prefix)
        if subresp .status < 300:
            print('Objects in folder [' + prefix.prefix + ']:')
            for content in subresp.body.contents:
                print('key:', content.key)
                print('owner_id:', content.owner.owner_id)
                print('owner_name:', content.owner.owner_name)
            listObjectsByPrefix(subresp)
        else:
            print('errorCode:', resp.errorCode)
            print('errorMessage:', resp.errorMessage)

resp = obsClient.listObjects('bucketname', delimiter='/')
if resp.status < 300:
    print('Objects in the root directory:')
    for content in resp.body.contents:
        print('key:', content.key)
        print('owner_id:', content.owner.owner_id)
        print('owner_name:', content.owner.owner_name)
    listObjectsByPrefix(resp)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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, **body.contents** includes the objects in the folder and **body.commonPrefixs** includes the sub-folders in the folder.

9.4 Deleting Objects

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='your-endpoint'
)

resp = obsClient.deleteObject('bucketname', 'objectname')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import DeleteObjectsRequest, Object
deleteObjectsRequest = DeleteObjectsRequest()
# Set the response mode to verbose.
deleteObjectsRequest.quiet = False
deleteObjectsRequest.objects = [Object(key='objectname1', versionId=None), Object(key='objectname2',
versionId=None)]
deleteObjectsRequest.objects.append(Object(key='objectname3', versionId=None))
```



```
resp = obsClient.deleteObjects('bucketname', deleteObjectsRequest=deleteObjectsRequest)
if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain the successfully deleted objects.
    if resp.body.deleted:
        index = 1
        for delete in resp.body.deleted:
            print('deleted[', index, ']:')
            print('key:', delete.key)
            print('versionId:', delete.versionId)
            index += 1
    # Obtain the list of objects failed to be deleted.
    if resp.body.error:
        index = 1
        for err in resp.body.error:
            print('error[', index, ']:')
            print('key:', err.key)
            print('code:', err.code)
            print('message:', err.message)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

9.5 Copying an Object

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.copyObject('sourcebucketname', 'sourceobjectname', 'destbucketname', 'destobjectname')

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Rewriting Object Properties

The following sample code shows how to rewrite object properties.

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
```



```
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import CopyObjectHeader
from obs import StorageClass
headers = CopyObjectHeader()
# Set the rewrite option of the object properties.
headers.directive = 'REPLACE'
headers.contentType = 'image/jpeg'
headers.storageClass = StorageClass.STANDARD
# Rewrite the customized metadata.
metadata = {'property' : 'property-value'}

resp = obsClient.copyObject('sourcebucketname', 'sourceobjectname', 'destbucketname', 'destobjectname',
headers=headers, metadata=metadata)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

Use the **metadata** parameter to specify the object's customized metadata to be rewritten and the **CopyObjectHeader.directive** parameter to specify the rewrite mode, which can be **REPLACE** (rewrite) or **COPY** (copy from the source object).

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 error code will be returned and the copy will fail.

You can set the following conditions:

Parameter	Description	Format
CopyObjectHeader.if_modified_since	Copies the source object if it is changed after the time specified by this parameter; otherwise, an error code is returned.	This parameter must conform to the HTTP time format specified in http://www.ietf.org/rfc/rfc2616.txt.
CopyObjectHeader.if_unmodified_since	Copies the source object if it is changed before the time specified by this parameter; otherwise, an error code is returned.	This parameter must conform to the HTTP time format specified in http://www.ietf.org/rfc/rfc2616.txt.
CopyObjectHeader.if_match	Copies the source object if its ETag is the same as the one specified by this parameter; otherwise, an error code is returned.	Character string

Parameter	Description	Format
CopyObjectHeader.if_none_match	Copies the source object if its ETag is different from the one specified by this parameter; otherwise, an error code is returned.	Character string

NOTE

- The ETag of the source object is the MD5 check value of the source object.
- If **if_unmodified_since**, **if_match**, **if_modified_since**, or **if_none_match** is included, and the specified condition is not met, error code **412 Precondition Failed**, will be returned.
- **if_modified_since** and **if_none_match** can be used together. So do **if_unmodified_since** and **if_match**.

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import CopyObjectHeader
headers = CopyObjectHeader()
headers.if_modified_since = 'Thu, 31 Dec 2015 16:00:00 GMT'
headers.if_none_match = 'none-match-etag'

resp = obsClient.copyObject('sourcebucketname', 'sourceobjectname', 'destbucketname', 'destobjectname',
headers=headers)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Rewriting an Object ACL

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import CopyObjectHeader
from obs import HeadPermission
headers = CopyObjectHeader()
# Rewrite the object ACL to public-read during the copy.
headers.acl = HeadPermission.PUBLIC_READ
```



```
resp = obsClient.copyObject('sourcebucketname', 'sourceobjectname', 'destbucketname', 'destobjectname',
headers=headers)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


10 Authorized Access

10.1 Using a URL for Authorized Access

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 Method	Special Operator (Sub-resource)	Bucket Name Required	Object Name Required
PUT Bucket	PUT	N/A	Yes	No
GET Buckets	GET	N/A	No	No
DELETE Bucket	DELETE	N/A	Yes	No
GET Objects	GET	N/A	Yes	No
GET Object versions	GET	versions	Yes	No
List Multipart uploads	GET	uploads	Yes	No
Obtain Bucket Metadata	HEAD	N/A	Yes	No
GET Bucket location	GET	location	Yes	No

Operation	HTTP Method	Special Operator (Sub-resource)	Bucket Name Required	Object Name Required
GET Bucket storageinfo	GET	storageinfo	Yes	No
PUT Bucket quota	PUT	quota	Yes	No
GET Bucket quota	GET	quota	Yes	No
Set Bucket storagePolicy	PUT	storagePolicy	Yes	No
GET Bucket storagePolicy	GET	storagePolicy	Yes	No
PUT Bucket acl	PUT	acl	Yes	No
GET Bucket acl	GET	acl	Yes	No
PUT Bucket logging	PUT	logging	Yes	No
GET Bucket logging	GET	logging	Yes	No
PUT Bucket policy	PUT	policy	Yes	No
GET Bucket policy	GET	policy	Yes	No
DELETE Bucket policy	DELETE	policy	Yes	No
PUT Bucket lifecycle	PUT	lifecycle	Yes	No
GET Bucket lifecycle	GET	lifecycle	Yes	No
DELETE Bucket lifecycle	DELETE	lifecycle	Yes	No
PUT Bucket website	PUT	website	Yes	No
GET Bucket website	GET	website	Yes	No
DELETE Bucket website	DELETE	website	Yes	No
PUT Bucket versioning	PUT	versioning	Yes	No

Operation	HTTP Method	Special Operator (Sub-resource)	Bucket Name Required	Object Name Required
GET Bucket versioning	GET	versioning	Yes	No
PUT Bucket cors	PUT	cors	Yes	No
GET Bucket cors	GET	cors	Yes	No
DELETE Bucket cors	DELETE	cors	Yes	No
PUT Bucket notification	PUT	notification	Yes	No
GET Bucket notification	GET	notification	Yes	No
PUT Bucket tagging	PUT	tagging	Yes	No
GET Bucket tagging	GET	tagging	Yes	No
DELETE Bucket tagging	DELETE	tagging	Yes	No
PUT Object	PUT	N/A	Yes	Yes
Append Object	POST	append	Yes	Yes
GET Object	GET	N/A	Yes	Yes
PUT Object - Copy	PUT	N/A	Yes	Yes
DELETE Object	DELETE	N/A	Yes	Yes
DELETE Objects	POST	delete	Yes	Yes
Obtain Object Metadata	HEAD	N/A	Yes	Yes
PUT Object acl	PUT	acl	Yes	Yes
GET Object acl	GET	acl	Yes	Yes
Initiate Multipart Upload	POST	uploads	Yes	Yes
PUT Part	PUT	N/A	Yes	Yes

Operation	HTTP Method	Special Operator (Sub-resource)	Bucket Name Required	Object Name Required
PUT Part - Copy	PUT	N/A	Yes	Yes
List Parts	GET	N/A	Yes	Yes
Complete Multipart Upload	POST	N/A	Yes	Yes
DELETE Multipart upload	DELETE	N/A	Yes	Yes
POST Object restore	POST	restore	Yes	Yes

To authorize a URL for access, perform the following two steps:

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

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

----End

The following code provides an example showing how to use a URL for authorized access by performing operations, including bucket creation, as well as object upload, download, listing, and deletion.

PUT Bucket

```
# Import the module.
from obs import ObsClient
import sys
IS_PYTHON2 = sys.version_info.major == 2 or sys.version < '3'

if IS_PYTHON2:
    from urlparse import urlparse
    import httplib
else:
    import http.client as httplib
    from urllib.parse import urlparse

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='http://your-endpoint'
)

bucketName = 'bucketname'
location = 'your location'
content = '<CreateBucketConfiguration><LocationConstraint>%s</LocationConstraint></
CreateBucketConfiguration>' % location

headers = {'Content-Length': str(len(content))}
```



```
res = obsClient.createSignedUrl('PUT', bucketName, expires=3600, headers=headers)
url = res['signedUrl']
print('Creating bucket using temporary signature url:')
print(res['signedUrl'])

url = urlparse(url)

# Make a PUT request to create a bucket.
conn = httplib.HTTPConnection(url.hostname, url.port)
path = url.path + '?' + url.query
conn.request('PUT', path, headers=res['actualSignedRequestHeaders'])

conn.send(content if IS_PYTHON2 else content.encode('UTF-8'))

result = conn.getresponse()
responseContent = result.read()

print('Response Status:')
print(result.status)

if responseContent:
    print('Response Content:')
    print('%s' % responseContent)
conn.close()
```

PUT Object

```
#!/usr/bin/python
# -*- coding:utf-8 -*-

# Import the module.
from obs import ObsClient
import sys
IS_PYTHON2 = sys.version_info.major == 2 or sys.version < '3'

if IS_PYTHON2:
    from urlparse import urlparse
    import httplib
else:
    import http.client as httplib
    from urllib.parse import urlparse

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='http://your-endpoint'
)

bucketName = 'bucketname'
objectKey = 'objectname'

content = 'Hello OBS'
headers = {'Content-Length': str(len(content))}

res = obsClient.createSignedUrl('PUT', bucketName, objectKey, expires=3600, headers=headers)
url = res['signedUrl']
print('Creating object using temporary signature url:')
print(res['signedUrl'])

url = urlparse(url)

# Make a PUT request to upload an object.
conn = httplib.HTTPConnection(url.hostname, url.port)
path = url.path + '?' + url.query
conn.request('PUT', path, headers=res['actualSignedRequestHeaders'])

conn.send(content if IS_PYTHON2 else content.encode('UTF-8'))
```



```
result = conn.getresponse()
responseContent = result.read()

print('Response Status:')
print(result.status)

if responseContent:
    print('Response Content:')
    print('%s' % responseContent)
conn.close()
```

GET Object

```
# Import the module.
from obs import ObsClient
import sys
IS_PYTHON2 = sys.version_info.major == 2 or sys.version < '3'

if IS_PYTHON2:
    from urlparse import urlparse
    import httplib
else:
    import http.client as httplib
    from urllib.parse import urlparse

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='http://your-endpoint'
)

bucketName = 'bucketname'
objectKey = 'objectname'

res = obsClient.createSignedUrl('GET', bucketName, objectKey, expires=3600)
url = res['signedUrl']
print('Getting object using temporary signature url:')
print(res['signedUrl'])

url = urlparse(url)

# Make a GET request to download an object.
conn = httplib.HTTPConnection(url.hostname, url.port)
path = url.path + '?' + url.query
conn.request('GET', path, headers=res['actualSignedRequestHeaders'])

result = conn.getresponse()
responseContent = result.read()

print('Response Status:')
print(result.status)

if responseContent:
    print('Response Content:')
    print('%s' % responseContent)
conn.close()
```

Listing Objects

```
# Import the module.
from obs import ObsClient
import sys
IS_PYTHON2 = sys.version_info.major == 2 or sys.version < '3'

if IS_PYTHON2:
    from urlparse import urlparse
    import httplib
else:
```



```
    import http.client as httplib
    from urllib.parse import urlparse

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='http://your-endpoint'
)

bucketName = 'bucketname'

res = obsClient.createSignedUrl('GET', bucketName, expires=3600)
url = res['signedUrl']
print('Listing object using temporary signature url:')
print(res['signedUrl'])

url = urlparse(url)

# Make a GET request to obtain the object list.
conn = httplib.HTTPConnection(url.hostname, url.port)
path = url.path + '?' + url.query
conn.request('GET', path, headers=res['actualSignedRequestHeaders'])

result = conn.getresponse()
responseContent = result.read()

print('Response Status:')
print(result.status)

if responseContent:
    print('Response Content:')
    print('%s' % responseContent)
conn.close()
```

DELETE Object

```
# Import the module.
from obs import ObsClient
import sys
IS_PYTHON2 = sys.version_info.major == 2 or sys.version < '3'

if IS_PYTHON2:
    from urlparse import urlparse
    import httplib
else:
    import http.client as httplib
    from urllib.parse import urlparse

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='http://your-endpoint'
)

bucketName = 'bucketname'
objectKey = 'objectname'

res = obsClient.createSignedUrl('DELETE', bucketName, objectKey, expires=3600)
url = res['signedUrl']
print('Getting object using temporary signature url:')
print(res['signedUrl'])

url = urlparse(url)

# Make a DELETE request to delete the object.
conn = httplib.HTTPConnection(url.hostname, url.port)
path = url.path + '?' + url.query
```



```
conn.request('DELETE', path, headers=res['actualSignedRequestHeaders'])

result = conn.getresponse()
responseContent = result.read()

print('Response Status:')
print(result.status)

if responseContent:
    print('Response Content:')
    print('%s' % responseContent)
conn.close()
```

11 Versioning Management

11.1 Versioning Overview

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**.

11.2 Setting Versioning Status for a Bucket

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

Versioning Status	Description
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. Objects of the latest version 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.

Versioning Status	Description
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 version ID. If an object is deleted with no version ID specified, the object is only attached with a deletion mark and version ID **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.

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Enable versioning.
resp = obsClient.setBucketVersioning('bucketname', 'Enabled')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)

# Suspend versioning.
resp = obsClient.setBucketVersioning('bucketname', 'Suspended')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

11.3 Viewing Versioning Status of a Bucket

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
```



```
)

resp = obsClient.getBucketVersioning('bucketname')
if resp.status < 300:
    print('requestId:', resp.requestId)
    print('status:', resp.body)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

11.4 Obtaining a Versioning Object

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import GetObjectRequest
# Set versionId to obtain a versioning object.
getObjectRequest = GetObjectRequest()
getObjectRequest.versionId = 'versionid'

resp = obsClient.getObject('bucketname', 'objectname', loadStreamInMemory=True,
getObjectRequest=getObjectRequest)
if resp.status < 300:
    print('requestId:', resp.requestId)
    print('buffer:', resp.body.buffer)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

If **versionId** is **None**, the object of the latest version will be downloaded, by default.

11.5 Copying an Object with a Specified Version ID

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

```
# Import the module.
from obs import ObsClient
from obs import RestoreTier

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Set the version ID of the object to be copied.
versionId = 'versionid'
# Restore a versioning object in the Expedited mode.
obsClient.copyObject('sourcebucketname', 'sourceobjectname', 'destbucketname', 'destobjectname',
versionId=versionId)
```



```
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

11.6 Restoring an Archive Object with a Specified Version ID

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

obsClient.restoreObject('bucketname', 'objectname', days=1, versionId='versionid',
tier=RestoreTier.EXPEDITED)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

11.7 Listing Versioning Objects

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

The following table describes the parameters involved in this API.

Parameter	Description
Versions.prefix	Name prefix that the objects to be listed must contain
Versions.key_marker	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.
Versions.max_keys	Maximum number of listed versioning objects. The value ranges from 1 to 1000. If the value is not in this range, 1000 versioning objects are returned by default.
Versions.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.)

Parameter	Description
Versions.version_id_marker	Object name to start with when listing versioning objects in a bucket. All versioning objects are listed in the lexicographical order by object name and version ID. This parameter must be used together with **key_marker**.

NOTE

- If the value of **version_id_marker** is not a version ID specified by **key_marker**, **version_id_marker** 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 listed.

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.listVersions('bucketname')
if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain versioning objects.
    index = 1
    for version in resp.body.versions:
        print('version [' + str(index) + ']')
        print('key:', version.key)
        print('versionId:', version.versionId)
        print('owner_id:', version.owner.owner_id)
        print('owner_name:', version.owner.owner_name)
        index += 1

    # Obtain delete markers.
    index = 1
    for marker in resp.body.markers:
        print('marker [' + str(index) + ']')
        print('key:', marker.key)
        print('versionId:', marker.versionId)
        print('owner_id:', marker.owner.owner_id)
        print('owner_name:', marker.owner.owner_name)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


NOTE

- A maximum of 1000 versioning objects can be listed each time. If a bucket contains more than 1000 objects and **body.head.isTruncated** is **True** in the returned result, not all versioning objects are listed. In such cases, you can use **body.head.nextKeyMarker** and **body.head.nextVersionIdMarker** 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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import Versions
versions = Versions()
# List 100 versioning objects.
versions.max_keys = 100

resp = obsClient.listVersions('bucketname', version=versions)

if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain versioning objects.
    index = 1
    for version in resp.body.versions:
        print('version [' + str(index) + ']')
        print('key:', version.key)
        print('versionId:', version.versionId)
        print('owner_id:', version.owner.owner_id)
        print('owner_name:', version.owner.owner_name)
        index += 1

    # Obtain delete markers.
    index = 1
    for marker in resp.body.markers:
        print('marker [' + str(index) + ']')
        print('key:', marker.key)
        print('versionId:', marker.versionId)
        print('owner_id:', marker.owner.owner_id)
        print('owner_name:', marker.owner.owner_name)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Listing Versioning Objects by Specifying a Prefix

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
```



```
    server='https://your-endpoint'
)

from obs import Versions
versions = Versions()
# Set the prefix to prefix and the number to 100.
versions.max_keys = 100
versions.prefix = 'prefix'

resp = obsClient.listVersions('bucketname', version=versions)

if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain versioning objects.
    index = 1
    for version in resp.body.versions:
        print('version [' + str(index) + ']')
        print('key:', version.key)
        print('versionId:', version.versionId)
        print('owner_id:', version.owner.owner_id)
        print('owner_name:', version.owner.owner_name)
        index += 1

    # Obtain delete markers.
    index = 1
    for marker in resp.body.markers:
        print('marker [' + str(index) + ']')
        print('key:', marker.key)
        print('versionId:', marker.versionId)
        print('owner_id:', marker.owner.owner_id)
        print('owner_name:', marker.owner.owner_name)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Listing Versioning Objects by Specifying the Start Position

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import Versions
versions = Versions()
# List 100 versioning objects whose names are following test in lexicographical order.
versions.max_keys = 100
versions.key_marker = 'test'

resp = obsClient.listVersions('bucketname', version=versions)

if resp.status < 300:
    print('requestId:', resp.requestId)
    # Obtain versioning objects.
    index = 1
    for version in resp.body.versions:
        print('version [' + str(index) + ']')
        print('key:', version.key)
        print('versionId:', version.versionId)
        print('owner_id:', version.owner.owner_id)
        print('owner_name:', version.owner.owner_name)
        index += 1
```



```
    # Obtain delete markers.
    index = 1
    for marker in resp.body.markers:
        print('marker [' + str(index) + ']')
        print('key:', marker.key)
        print('versionId:', marker.versionId)
        print('owner_id:', marker.owner.owner_id)
        print('owner_name:', marker.owner.owner_name)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Listing All Versioning Objects in Paging Mode

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import Versions
versions = Versions()
versions.max_keys = 100

while True:
    resp = obsClient.listVersions('bucketname', version=versions)
    if resp.status < 300:
        print('requestId:', resp.requestId)
        # Obtain versioning objects.
        index = 1
        for version in resp.body.versions:
            print('version [' + str(index) + ']')
            print('key:', version.key)
            print('versionId:', version.versionId)
            print('owner_id:', version.owner.owner_id)
            print('owner_name:', version.owner.owner_name)
            index += 1
        # Obtain delete markers.
        index = 1
        for marker in resp.body.markers:
            print('marker [' + str(index) + ']')
            print('key:', marker.key)
            print('versionId:', marker.versionId)
            print('owner_id:', marker.owner.owner_id)
            print('owner_name:', marker.owner.owner_name)
            index += 1

        if not resp.body.head.isTruncated:
            break
        versions.key_marker = resp.body.head.nextKeyMarker
        versions.version_id_marker = resp.body.head.nextVersionIdMarker
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        break
```

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import Versions
versions = Versions()
versions.max_keys = 100
versions.prefix = 'dir/'

while True:
    resp = obsClient.listVersions('bucketname', version=versions)
    if resp.status < 300:
        print('requestId:', resp.requestId)
        # Obtain versioning objects.
        index = 1
        for version in resp.body.versions:
            print('version [' + str(index) + ']')
            print('key:', version.key)
            print('versionId:', version.versionId)
            print('owner_id:', version.owner.owner_id)
            print('owner_name:', version.owner.owner_name)
            index += 1
        # Obtain delete markers.
        index = 1
        for marker in resp.body.markers:
            print('marker [' + str(index) + ']')
            print('key:', marker.key)
            print('versionId:', marker.versionId)
            print('owner_id:', marker.owner.owner_id)
            print('owner_name:', marker.owner.owner_name)
            index += 1
        if not resp.body.head.isTruncated:
            break
        versions.key_marker = resp.body.head.nextKeyMarker
        versions.version_id_marker = resp.body.head.nextVersionIdMarker
    else:
        print('errorCode:', resp.errorCode)
        print('errorMessage:', resp.errorMessage)
        break
```

Listing All Versioning Objects According to Folders in a Bucket

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

def listVersionsByPrefix(resp, versions):
    for prefix in resp.body.commonPrefixs:
        versions.prefix = prefix.prefix
        subresp = obsClient.listVersions('bucketname', version=versions)
        if subresp.status < 300:
```



```
            print('Objects in folder [' + prefix.prefix + ']:')
            # Obtain versioning objects.
            index = 1
            for version in subresp.body.versions:
                print('version [' + str(index) + ']')
                print('key:', version.key)
                print('versionId:', version.versionId)
                print('owner_id:', version.owner.owner_id)
                print('owner_name:', version.owner.owner_name)
                index += 1
        # Obtain delete markers.
            index = 1
            for marker in subresp.body.markers:
                print('marker [' + str(index) + ']')
                print('key:', marker.key)
                print('versionId:', marker.versionId)
                print('owner_id:', marker.owner.owner_id)
                print('owner_name:', marker.owner.owner_name)
                index += 1
        else:
            print('errorCode:', resp.errorCode)
            print('errorMessage:', resp.errorMessage)

from obs import Versions
versions = Versions()
versions.delimiter = '/'

resp = obsClient.listVersions('bucketname', version=versions)

if resp.status < 300:
    print('Objects in the root directory:')
    # Obtain versioning objects.
    index = 1
    for version in resp.body.versions:
        print('version [' + str(index) + ']')
        print('key:', version.key)
        print('versionId:', version.versionId)
        print('owner_id:', version.owner.owner_id)
        print('owner_name:', version.owner.owner_name)
        index += 1
        # Obtain delete markers.
    index = 1
    for marker in resp.body.markers:
        print('marker [' + str(index) + ']')
        print('key:', marker.key)
        print('versionId:', marker.versionId)
        print('owner_id:', marker.owner.owner_id)
        print('owner_name:', marker.owner.owner_name)
        index += 1
    listVersionsByPrefix(resp)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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, **body.versions** includes the versioning objects in the folder, **body.markers** includes the delete markers in the folder, and **body.commonPrefixs** includes the sub-folders in the folder.

11.8 Setting or Obtaining a Versioning Object ACL

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:

```
# Import the module.
from obs import ObsClient
from obs import HeadPermission

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Set the versioning object ACL to public-read by specifying the pre-defined access control policy.
resp = obsClient.setObjectAcl('bucketname', 'objectname', aclControl=HeadPermission.PUBLIC_READ,
versionId='versionid')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)

from obs import ACL
from obs import Owner
from obs import Grant, Permission
from obs import Grantee, Group
owner = Owner(owner_id='ownerid')

# Grant all permissions to a specified user.
grant0 = Grant(grantee=Grantee(grantee_id='userid'), permission=Permission.FULL_CONTROL)

# Grant the READ permission to all users.
grant1 = Grant(grantee=Grantee(group=Group.ALL_USERS), permission=Permission.READ)

acl = ACL(owner=owner, grants=[grant0, grant1])

# Set ACL for a specified versioning object.
resp = obsClient.setObjectAcl('bucketname', 'objectname', acl=acl, versionId='versionid')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
```

```
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.getObjectAcl('bucketname', 'objectname', versionId='versionid')
if resp.status < 300:
    print('requestId:', resp.requestId)
    print('owner_id:', resp.body.owner.owner_id)
    print('owner_name:', resp.body.owner.owner_name)
    index = 1
    for grant in resp.body.grants:
        print('grant [' + str(index) + ']')
        print('grantee_id:', grant.grantee.grantee_id)
        print('grantee_name:', grant.grantee.grantee_name)
        print('group:', grant.grantee.group)
        print('permission:', grant.permission)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

11.9 Deleting Versioning Objects

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.deleteObject('bucketname', 'objectname', versionId='versionid')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Deleting Versioning Objects in a Batch

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import DeleteObjectsRequest, Object

object1 = Object(key='objectname1', versionId='version1')
```



```
object2 = Object(key='objectname2', versionId='version2')

resp = obsClient.deleteObjects('bucketname', DeleteObjectsRequest(quiet=False, objects=[object1, object2]))

if resp.status < 300:
    print('requestId:', resp.requestId)
    if resp.body.deleted:
        index = 1
        for delete in resp.body.deleted:
            print('delete[' + str(index) + ']')
            print('key:', delete.key, ',deleteMarker:', delete.deleteMarker, ',deleteMarkerVersionId:',
delete.deleteMarkerVersionId)
            print('versionId:', delete.versionId)
            index += 1
    if resp.body.error:
        index = 1
        for err in resp.body.error:
            print('err[' + str(index) + ']')
            print('key:', err.key, ',code:', err.code, ',message:', err.message)
            print('versionId:', err.versionId)
            index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


12 Lifecycle Management

12.1 Lifecycle Management Overview

OBS allows you to set lifecycle rules for buckets to automatically transit the storage class of objects and delete expired objects, so as to effectively use storage features and optimize the storage space. You can set multiple lifecycle rules to match different object name prefixes. A lifecycle rule must contain:

- Rule ID, which uniquely identifies the rule
- A shared prefix of object names 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.

12.2 Setting Lifecycle Rules

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

Setting an Object Transition Policy

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import Transition, NoncurrentVersionTransition
from obs import DateTime
from obs import Rule
from obs import Lifecycle
from obs import StorageClass
# Specify that objects whose names contain the prefix will be transited to OBS Infrequent Access 30 days after creation, and that objects whose names contain the prefix will be transited to OBS Archive 30 days after changing into noncurrent versions.
rule1 = Rule(id='rule1', prefix='prefix1', status='Enabled',
transition=Transition(storageClass=StorageClass.WARM, days=30),
noncurrentVersionTransition=NoncurrentVersionTransition(storageClass=StorageClass.COLD,
noncurrentDays=30))

# Directly specify the date when objects whose name contain the prefix will be transited to OBS Infrequent Access.
rule2 = Rule(id='rule2', prefix='prefix2', status='Enabled',
transition=Transition(storageClass=StorageClass.WARM, date=DateTime(2018, 10, 31)))

lifecycle = Lifecycle(rule=[rule1, rule2])

resp = obsClient.setBucketLifecycle('bucketname', lifecycle)
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Setting an Object Expiration Time

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
```

```
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import Expiration, NoncurrentVersionExpiration
from obs import DateTime
from obs import Rule
from obs import Lifecycle
# Specify that objects whose names contain the prefix will expire 60 days after creation, and that objects whose names contain the prefix will expire 60 days after changing into noncurrent versions.
rule1 = Rule(id='rule1', prefix='prefix1', status='Enabled', expiration=Expiration(days=60), noncurrentVersionExpiration=NoncurrentVersionExpiration(noncurrentDays=60))

# Specify a date when the objects whose names contain the prefix will expire.
rule2 = Rule(id='rule2', prefix='prefix2', status='Enabled', expiration=Expiration(date=DateTime(2018, 12, 31)))

lifecycle = Lifecycle(rule=[rule1, rule2])

resp = obsClient.setBucketLifecycle('bucketname', lifecycle)
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

12.3 Viewing Lifecycle Rules

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.getBucketLifecycle('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
    index = 1
    for rule in resp.body.lifecycleConfig.rule:
        print('rule [' + str(index) + ']')
        print('id:', rule.id)
        print('prefix:', rule.prefix)
        print('status:', rule.status)
        if rule.transition:
            print('days:' , rule.transition.days)
            print('date:' , rule.transition.date)
            print('storageClass:' , rule.transition.storageClass)
        if rule.expiration:
            print('days:' , rule.expiration.days)
            print('date:' , rule.expiration.date)
        if rule.noncurrentVersionTransition:
            print('noncurrentDays:' , rule.noncurrentVersionTransition.noncurrentDays)
            print('storageClass:' , rule.noncurrentVersionTransition.storageClass)
        if rule.noncurrentVersionExpiration:
            print('noncurrentDays:' , rule.noncurrentVersionExpiration.noncurrentDays)
    index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


12.4 Deleting Lifecycle Rules

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.deleteBucketLifecycle('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


13 CORS

13.1 CORS Overview

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**.

13.2 Setting CORS Rules

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import CorsRule

# Specify the request method, which can be GET, PUT, DELETE, POST, or HEAD.
allowedMethod = ['PUT', 'POST', 'GET', 'DELETE', 'HEAD']
# Specify the origin of the cross-origin request.
allowedOrigin = ['http://www.a.com', 'http://www.b.com']
# Specify whether headers specified in Access-Control-Request-Headers in the OPTIONS request can be used.
allowedHeader = ['x-obs-header']
# Specify the browser's cache time of the returned results of OPTIONS requests for specific resources, in seconds.
maxAgeSecond = 60
# Specify response headers that users can access using application programs.
exposeHeader = ['x-obs-expose-header']
cors1 = CorsRule(id='rule1', allowedMethod=allowedMethod,
                 allowedOrigin=allowedOrigin, allowedHeader=allowedHeader,
                 maxAgeSecond=maxAgeSecond, exposeHeader=exposeHeader)

resp = obsClient.setBucketCors('bucketname', corsList=[cors1])
```



```
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

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

13.3 Viewing CORS Rules

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.getBucketCors('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
    index = 1
    for rule in resp.body:
        print('corsRule [' + str(index) + ']')
        print('id:', rule.id)
        print('allowedMethod', rule.allowedMethod)
        print('allowedOrigin', rule.allowedOrigin)
        print('allowedHeader', rule.allowedHeader)
        print('maxAgeSecond', rule.maxAgeSecond)
        print('exposeHeader', rule.exposeHeader)
        index +=1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

13.4 Deleting CORS Rules

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.deleteBucketCors('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


14 Access Logging

14.1 Logging Overview

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**.

14.2 Enabling Bucket Logging

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:

```
# Import the module.
from obs import ObsClient
from obs import HeadPermission
from obs import Logging

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
```



```
logstatus = Logging(targetBucket='targetbucketname', targetPrefix='prefix', agency='your agency')
    # Configure logging for the bucket.
resp = obsClient.setBucketLogging('bucketname', logstatus)
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Setting ACLs for Objects to Be Logged

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import Grantee, Group
from obs import Grant, Permission
from obs import Logging

    # Grant the READ permission on the objects to be logged to all users.
grantee1 = Grantee(group=Group.ALL_USERE)
grant1 = Grant(grantee=grantee1, permission=Permission.READ)
logstatus = Logging(targetBucket='targetbucketname', targetPrefix='prefix', agency='your
agency',targetGrants= [grant1])

    # Configure logging for the bucket.
resp = obsClient.setBucketLogging('bucketname', logstatus)
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

14.3 Viewing Bucket Logging

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.getBucketLogging('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
    print('targetBucket:', resp.body.targetBucket)
    print('targetPrefix:', resp.body.targetPrefix)

    index = 1
    for grant in resp.body.targetGrants:
        print('grant [' + str(index) + ']')
```



```
        print('grant_id:', grant.grantee.grantee_id)
        print('grant_name:', grant.grantee.grantee_name)
        print('group:', grant.grantee.group)
        print('permission:', grant.permission)
        index += 1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

14.4 Disabling Bucket Logging

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import Logging
 # Leave the logging settings in blank.
resp = obsClient.setBucketLogging('bucketname', Logging())
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


15 Static Website Hosting

15.1 Static Website Hosting Overview

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**.

15.2 Website File Hosting

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:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import PutObjectHeader
from obs import HeadPermission
headers = PutObjectHeader()
```



```
# Set the MIME type for the object.
headers.contentType = 'text/html'

# Upload an object.
resp = obsClient.putFile('bucketname', 'test.html', 'localfile.html', headers=headers)
if resp.status < 300:
    print('requestId:', resp.requestId)
    # Set the object ACL to public-read.
    resp2 = obsClient.setObjectAcl('bucketname', 'test.html', aclControl=HeadPermission.PUBLIC_READ)
    if resp2.status < 300:
        print('requestId:', resp2.requestId)
    else:
        print('errorCode:', resp2.errorCode)
        print('errorMessage:', resp2.errorMessage)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

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

15.3 Setting Website Hosting

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

Configuring the Default Homepage and Error Pages

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import WebsiteConfiguration
from obs import IndexDocument
from obs import ErrorDocument
from obs import RoutingRule
from obs import Condition
from obs import Redirect

# Configure the error pages.
errorDocument = ErrorDocument(key='error.html')
# Configure the default homepage.
indexDocument = IndexDocument(suffix='index.html')
resp = obsClient.setBucketWebsite('bucketname',
                WebsiteConfiguration(errorDocument=errorDocument, indexDocument=indexDocument))

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Configuring Redirection Rules

Sample code:


```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import WebsiteConfiguration
from obs import IndexDocument
from obs import ErrorDocument
from obs import RoutingRule
from obs import Condition
from obs import Redirect

#  Configure the error pages.
errorDocument = ErrorDocument(key='error.html')
# Configure the default homepage.
indexDocument = IndexDocument(suffix='index.html')
routingRule = RoutingRule(condition=Condition(keyPrefixEquals='keyprefix'),
                          redirect=Redirect(protocol='http', replaceKeyPrefixWith='replacekeyprefix',
httpRedirectCode=305, hostName='www.example.com'))
resp = obsClient.setBucketWebsite('bucketname',
                WebsiteConfiguration(errorDocument=errorDocument, indexDocument=indexDocument,
routingRules=[routingRule]))

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Configuring Redirection for All Requests

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import WebsiteConfiguration
from obs import RedirectAllRequestTo

resp = obsClient.setBucketWebsite('bucketname',

WebsiteConfiguration(redirectAllRequestTo=RedirectAllRequestTo(hostName='www.example.com',
protocol='http')))

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

15.4 Viewing Website Hosting Settings

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.getBucketWebsite('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
    if resp.body.redirectAllRequestTo:
        print('redirectAllRequestTo.hostName:', resp.body.redirectAllRequestTo.hostName,
',redirectAllRequestTo.protocol:', resp.body.redirectAllRequestTo.protocol)
    if resp.body.indexDocument:
        print('indexDocument.suffix:', resp.body.indexDocument.suffix)
    if resp.body.errorDocument:
        print('errorDocument.key:', resp.body.errorDocument.key)
    if resp.body.routingRules:
        index = 1
        for rout in resp.body.routingRules:
            print('routingRule[', index, ']:')
            index += 1
            print('condition.keyPrefixEquals:', rout.condition.keyPrefixEquals,
',condition.httpErrorCodeReturnedEquals:', rout.condition.httpErrorCodeReturnedEquals)
            print('redirect.protocol:', rout.redirect.protocol, ',redirect.hostName:', rout.redirect.hostName,
',redirect.replaceKeyPrefixWith:', rout.redirect.replaceKeyPrefixWith, ',redirect.replaceKeyWith:',
rout.redirect.replaceKeyWith, ',redirect.httpRedirectCode:', rout.redirect.httpRedirectCode)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

15.5 Deleting Website Hosting Settings

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.deleteBucketWebsite('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


16 Tag Management

16.1 Tagging Overview

Tags are used to identify and classify OBS buckets.

For more information, see **Tags**.

16.2 Setting Bucket Tags

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import TagInfo

tagInfo = TagInfo()
tagInfo.addTag('tag1', 'value1').addTag('tag2', 'value2')

resp = obsClient.setBucketTagging('bucketname', tagInfo)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

NOTE

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

16.3 Viewing Bucket Tags

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.getBucketTagging('bucketname')
if resp.status < 300:
    print('requestId:', resp.requestId)
    index = 1
    for tag in resp.body.tagSet:
        print('tag [' + str(index) + ']')
        print('key:', tag.key)
        print('value:', tag.value)
        index +=1
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

16.4 Deleting Bucket Tags

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

resp = obsClient.deleteBucketTagging('bucketname')
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


17 Event Notification

17.1 Event Notification Overview

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**.

17.2 Setting Event Notification

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import Notification, TopicConfiguration, FilterRule
from obs import EventType

fr1 = FilterRule(name='prefix', value='smn')
fr2 = FilterRule(name='suffix', value='.jpg')
topicConfiguration = TopicConfiguration(id='001',topic='your
topic',events=[EventType.OBJECT_CREATED_ALL], filterRules=[fr1,fr2])

fr3 = FilterRule(name='prefix', value='function')
fr4 = FilterRule(name='suffix', value='.mp4')
functionConfiguration = FunctionGraphConfiguration(id='002', functionGraph='your function',
events=[EventType.OBJECT_CREATED_ALL], filterRules=[fr3, fr4])

resp = obsClient.setBucketNotification('bucketname', Notification(topicConfigurations=[topicConfiguration],
functionGraphConfigurations=[functionConfiguration]))

if resp.status < 300:
    print('requestId:', resp.requestId)
```



```
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

17.3 Viewing Event Notification Settings

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

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.getBucketNotification('bucketname')

if resp.status < 300:
    print('requestId:', resp.requestId)
    for topicConfiguration in resp.body.topicConfigurations:
        print('id:', topicConfiguration.id)
        print('topic:', topicConfiguration.topic)
        print('events:', topicConfiguration.events)
        index = 1
        for rule in topicConfiguration.filterRules:
            print('rule [' + str(index) + ']')
            print('name:', rule.name)
            print('value:', rule.value)

    for functionGraphConfiguration in resp.body.functionGraphConfigurations:
        print('id:', functionGraphConfiguration.id)
        print('functionGraph:', functionGraphConfiguration.functionGraph)
        print('events:', functionGraphConfiguration.events)
        index = 1
        for rule in functionGraphConfiguration.filterRules:
            print('rule [' + str(index) + ']')
            print('name:', rule.name)
            print('value:', rule.value)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

17.4 Disabling Event Notification

To disable event notification on buckets is to call **ObsClient.setBucketNotification** to clear all event notification settings. Sample code is as follows:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
from obs import Notification

resp = obsClient.setBucketNotification('bucketname', Notification())
if resp.status < 300:
```



```
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


18 Server-Side Encryption

18.1 Server-Side Encryption Overview

OBS supports server-side encryption.

For more information, see **Server-Side Encryption**.

18.2 Encryption Description

The following table lists APIs related to server-side encryption:

Method in OBS Python SDK	Description	Supported Encryption Type
ObsClient.putContent	Sets the encryption algorithm and key during object upload to enable server-side encryption.	SSE-KMS SSE-C
ObsClient.putFile	Sets the encryption algorithm and key during file upload to enable server-side encryption.	SSE-KMS SSE-C
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

Method in OBS Python SDK	Description	Supported Encryption Type
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 object copy. 2. Sets the encryption algorithm and key during object copy to enable the encryption algorithm for the target part.	SSE-C

18.3 Example of Encryption

Encrypting an Object to Be Uploaded

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import PutObjectHeader
from obs import SseCHeader, SseKmsHeader
headers = PutObjectHeader()
# Set the SSE-C encryption algorithm.
headers.sseHeader = SseCHeader(encryption='AES256', key='your sse-c key generated by AES-256
algorithm')

resp = obsClient.putFile('bucketname', 'objectname', 'localfile', headers=headers)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)

headers = PutObjectHeader()
# Set the SSE-KMS encryption algorithm.
```



```
headers.sseHeader = SseKmsHeader.getInstance()
resp = obsClient.putFile('bucketname', 'objectname2', 'localfile2', headers=headers2)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```

Decrypting a Downloaded Object

Sample code:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

from obs import GetObjectHeader
from obs import SseCHeader
headers = GetObjectHeader()
# Set the SSE-C decryption algorithm. The key used here must be the one used for uploading the object.
headers.sseHeader = SseCHeader(encryption='AES256', key='your sse-c key generated by AES-256
algorithm')
resp = obsClient.getObject('bucketname', 'objectname', 'localfile', headers=headers)

if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


19 Troubleshooting

19.1 HTTP Status Codes

The OBS server complies with the HTTP standard. After an API is called, the OBS server returns a standard HTTP status code. The following tables list the categories of HTTP status codes and the common HTTP status codes in OBS.

- Categories of HTTP status codes

Category	Description
1XX	Informational response. A request is received by the server and the server requires the requester to continue the operation. This category is usually invisible to the client.
2XX	Success. The operation is received and processed successfully.
3XX	Redirection. Further operations to complete the request are required.
4XX	Client errors. The request contains a syntax error, or the request cannot be implemented.
5XX	Server errors. An error occurs when the server is processing the request.

- Common HTTP status codes in OBS and their meanings

HTTP Status Code	Description	Possible Cause
400 Bad Request	Incorrect request parameter.	• Invalid request parameters. • The consistency check fails after the client request carries MD5. • An invalid parameter is transferred when the SDK is used. • An invalid bucket name is used.
403 Forbidden	Access denied.	• The signature of the request does not match. Generally, the error is caused by incorrect AK/SK. • The account does not have the permission to access the requested resource. • The account is in arrears. • The bucket space is insufficient when a quota is set for the bucket. • Invalid AK • The time difference between the client and the server is too large. That is, the time of the server where the client is located is not synchronized with the time of the NTP service.
404 Not Found	The requested resource does not exist.	• The bucket does not exist. • The object does not exist. • The bucket policy configuration does not exist. For example, the bucket CORS configuration or bucket policy configuration does not exist. • The multipart upload does not exist.
405 Method Not Allowed	The request method is not supported.	The requested method or feature is not supported in the region where the bucket resides.

HTTP Status Code	Description	Possible Cause
408 Request Timeout	Request timed out.	The Socket connection between the server and client timed out.
409 Conflict	Request conflicts occur.	• Buckets of the same name are created in different regions. • Deletion of a non-empty bucket is attempted.
500 Internal Server Error	Internal server error.	An internal error occurs on the server side.
503 Service Unavaliable	Service unavailable.	The server cannot be accessed temporarily.

19.2 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.

HTTP Status Code	Error Code	Error Message	Solution
301 Moved Permanently	PermanentRedirect	The requested bucket can be accessed only through the specified address. Send subsequent requests to the address.	Send the request to the returned redirection address.
301 Moved Permanently	WebsiteRedirect	The website request lacks **bucketName**.	Put the bucket name in the request and try again.
307 Moved Temporarily	TemporaryRedirect	Temporary redirection. If the DNS is updated, the request is redirected to the bucket.	The system automatically redirects the request or sends the request to the redirection address.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	BadDigest	The specified value of **Content-MD5** does not match the value received by OBS.	Check whether the MD5 value carried in the header is the same as that calculated by the message body.
400 Bad Request	BadDomainName	Invalid domain name.	Use a valid domain name.
400 Bad Request	BadRequest	Invalid request parameters.	Modify the parameter according to the error details returned in the message body.
400 Bad Request	CustomDomainAreadyExist	The configured domain already exists.	It has been configured and does not need to be configured again.
400 Bad Request	CustomDomainNotExist	The domain to be deleted does not exist.	The domain is not configured or has been deleted. You do not need to delete it.
400 Bad Request	EntityTooLarge	The size of the object uploaded using the POST method exceeds the upper limit.	Modify the conditions specified in the policy when posting the object or reduce the object size.
400 Bad Request	EntityTooSmall	The size of the object uploaded using the POST method does not reach the lower limit.	Modify the conditions specified in the policy when posting the object or increase the object size.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	IllegalLocation-ConstraintException	A request without **Location** is sent for creating a bucket in a non-default region.	Send the bucket creation request to the default region, or send the request with the **Location** of the non-default region.
400 Bad Request	IncompleteBody	No complete request body is received due to network or other problems.	Upload the object again.
400 Bad Request	IncorrectNumber-OfFilesInPost Request	Each POST request must contain one file to be uploaded.	Carry a file to be uploaded.
400 Bad Request	InvalidArgument	Invalid parameter.	Modify the parameter according to the error details returned in the message body.
400 Bad Request	InvalidBucket	The bucket to be accessed does not exist.	Try another bucket name.
400 Bad Request	InvalidBucketNam e	The bucket name specified in the request is invalid, which may have exceeded the maximum length, or contain special characters that are not allowed.	Try another bucket name.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	InvalidEncryptio-nAlgorithmError	Incorrect encryption algorithm. The object cannot be decrypted due to incorrect encryption header carried when downloading the SSE-C encrypted object.	Carry the correct encryption header when downloading the object.
400 Bad Request	InvalidLocation-Constraint	The specified **Location** in the bucket creation request is invalid or does not exist.	Correct the **Location** in the bucket creation request.
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.	Merge the parts correctly according to the ETags.
400 Bad Request	InvalidPartOrder	Parts are not listed in ascending order by part number.	Sort the parts in ascending order and merge them again.
400 Bad Request	InvalidPolicyDocu-ment	The content of the form does not meet the conditions specified in the policy document.	Modify the policy in the constructed form according to the error details in the message body and try again.
400 Bad Request	InvalidRedirectLo-cation	Invalid redirect location.	Specify the correct IP address.
400 Bad Request	InvalidRequest	Invalid request.	Modify the parameter according to the error details returned in the message body.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	InvalidRequestBody	The request body is invalid. The request requires a message body but no message body is uploaded.	Upload the message body in the correct format.
400 Bad Request	InvalidTargetBucketForLogging	The delivery group has no ACL permission for the target bucket.	Configure the target bucket ACL and try again.
400 Bad Request	KeyTooLongError	The provided key is too long.	Use a shorter key.
400 Bad Request	KMS.DisabledException	The master key is disabled in server-side encryption with KMS-managed keys (SSE-KMS) mode.	Replace the key and try again, or contact with the technical support.
400 Bad Request	KMS.NotFoundException	The master key does not exist in SSE-KMS mode.	Retry with the correct master key.
400 Bad Request	MalformedACLError	The provided XML file is in an incorrect format or does not meet format requirements.	Use the correct XML format to retry.
400 Bad Request	MalformedError	The XML format in the request is incorrect.	Use the correct XML format to retry.
400 Bad Request	MalformedLoggingStatus	The XML format of **Logging** is incorrect.	Use the correct XML format to retry.
400 Bad Request	MalformedPolicy	The bucket policy failed the check.	Modify the bucket policy according to the error details returned in the message body.
400 Bad Request	MalformedQuotaError	The Quota XML format is incorrect.	Use the correct XML format to retry.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	MalformedXML	An XML file of a configuration item is in incorrect format.	Use the correct XML format to retry.
400 Bad Request	MaxMessageLengthExceeded	Copying an object does not require a message body in the request.	Remove the message body and retry.
400 Bad Request	MetadataTooLarge	The size of the metadata header has exceeded the upper limit.	Reduce the size of the metadata header.
400 Bad Request	MissingRegion	No region contained in the request and no default region defined in the system.	Carry the region information in the request.
400 Bad Request	MissingRequestBodyError	An empty XML file is sent as a request.	Provide the correct XML file.
400 Bad Request	MissingRequired-Header	A required header is missing in the request.	Provide the required header.
400 Bad Request	MissingSecurity-Header	A required header is missing in the request.	Provide the required header.
400 Bad Request	TooManyBuckets	You have attempted to create more buckets than allowed.	Delete some buckets and try again.
400 Bad Request	TooManyCustomDomains	Too many user accounts are configured.	Delete some user accounts and try again.
400 Bad Request	TooManyWrongSignature	The request is rejected due to high-frequency errors.	Replace AK and try again.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	UnexpectedContent	The request requires a message body which is not carried by the client, or the request does not require a message body but the client carries the message body.	Try again according to the instruction.
400 Bad Request	UserKeyMustBeSpecified	This operation is only available to special users.	Contact the technical support.
403 Forbidden	AccessDenied	Access denied, because the request does not carry a date header or the header format is incorrect.	Provide a correct date header in the request.
403 Forbidden	AccessForbidden	Insufficient permission. No CORS rule is configured for the bucket or the CORS rule does not match.	Modify the CORS configuration of the bucket or send the matched OPTIONS request based on the CORS configuration of the bucket.
403 Forbidden	AllAccessDisabled	You have no permission to perform the operation. The bucket name is forbidden.	Try another bucket name.
403 Forbidden	DeregisterUserId	The user has been deregistered.	Top up or re-register.

HTTP Status Code	Error Code	Error Message	Solution
403 Forbidden	InArrearOrInsufficientBalance	The subscriber owes fees or the account balance is insufficient, and the subscriber does not have the permission to perform an operation.	Top up the account.
403 Forbidden	InsufficientStorageSpace	Insufficient storage space.	If the quota is exceeded, increase quota or delete some objects.
403 Forbidden	InvalidAccessKeyId	The access key ID provided by the customer does not exist in the system.	Provide correct access key ID.
403 Forbidden	NotSignedUp	Your account has not been registered with the system. Only a registered account can be used.	Register OBS.
403 Forbidden	RequestTimeTooSkewed	The request time and the server's time differ a lot.	Check whether the difference between the client time and the current time is too large.
403 Forbidden	SignatureDoesNotMatch	The provided signature in the request does not match the signature calculated by OBS.	Check your secret access key and signature calculation method.
403 Forbidden	Unauthorized	You have not been authenticated in real name.	Authenticate your real name and try again.
404 Not Found	NoSuchBucket	The specified bucket does not exist.	Create a bucket and perform the operation again.

HTTP Status Code	Error Code	Error Message	Solution
404 Not Found	NoSuchBucketPolicy	No bucket policy exists.	Configure a bucket policy.
404 Not Found	NoSuchCORSConfiguration	No CORS configuration exists.	Configure CORS first.
404 Not Found	NoSuchCustomDomain	The requested user domain does not exist.	Set a user domain first.
404 Not Found	NoSuchKey	The specified key does not exist.	Upload the object first.
404 Not Found	NoSuchLifecycle-Configuration	The requested lifecycle rule does not exist.	Configure a lifecycle rule first.
404 Not Found	NoSuchUpload	The specified multipart upload does not exist. The upload ID does not exist or the multipart upload job has been aborted or completed.	Use the existing part or reinitialize the part.
404 Not Found	NoSuchVersion	The specified version ID does not match any existing version.	Use a correct version ID.
404 Not Found	NoSuchWebsiteConfiguration	The requested website does not exist.	Configure the website first.
405 Method Not Allowed	MethodNotAllowed	The specified method is not allowed against the requested resource.	The method is not allowed.
408 Request Timeout	RequestTimeout	No read or write operation has been performed within the timeout period of the socket connection between the user and the server.	Check the network and try again, or contact technical support.

HTTP Status Code	Error Code	Error Message	Solution
409 Conflict	BucketAlreadyExists	The requested bucket name already exists. The bucket namespace is shared by all users of OBS. Select another name and retry.	Try another bucket name.
409 Conflict	BucketAlreadyOwnedByYou	Your previous request for creating the named bucket succeeded and you already own it.	You do not need to create the bucket again.
409 Conflict	BucketNotEmpty	The bucket that you tried to delete is not empty.	Delete the objects in the bucket and then delete the bucket.
409 Conflict	InvalidBucketState	Invalid bucket status. After cross-region replication is configured, bucket versioning cannot be disabled.	Enable bucket versioning or cancel cross-region replication.
409 Conflict	OperationAborted	A conflicting operation is being performed on this resource. Retry later.	Try again later.
409 Conflict	ServiceNotSupported	The request method is not supported by the server.	Not supported by the server. Contact the technical support.
411 Length Required	MissingContentLength	The HTTP header Content-Length is not provided.	Provide the Content-Length header.
412 Precondition Failed	PreconditionFailed	At least one of the specified preconditions is not met.	Modify according to the condition prompt in the returned message body.

HTTP Status Code	Error Code	Error Message	Solution
416 Client Requested Range Not Satisfiable	InvalidRange	The requested range cannot be obtained.	Retry with the correct range.
500 Internal Server Error	InternalError	An internal error occurs. Retry later.	Contact the technical support.
501 Not Implemented	ServiceNotImplemented	The request method is not implemented by the server.	Not supported currently. Contact the technical support.
503 Service Unavailable	ServiceUnavailable	The server is overloaded or has internal errors.	Try again later or contact the technical support.
503 Service Unavailable	SlowDown	Too frequent requests. Reduce your request frequency.	Too frequent requests. Reduce your request frequency.

19.3 SDK Common Result Objects

Each time an ObsClient related API is called (excluding **ObsClient.createSignedUrl** and **ObsClient.createPostSignature**), an SDK common result object will be returned. You can obtain the HTTP status code from the result to check whether the operation is successful. The object contains the following fields:

Field	Type	Description
status	int	HTTP status code. If the value is smaller than **300**, the operation succeeds. Otherwise, the operation fails.
reason	str	Reason description
errorCode	str	Error code returned by the OBS server. If the value of **status** is smaller than **300**, the value of this field is **None**.
errorMessage	str	Error description returned by the OBS server. If the value of **status** is smaller than **300**, the value of this field is **None**.
requestId	str	Request ID returned by the OBS server.
indicator	str	Error indicator returned by the OBS server

Field	Type	Description
hostId	str	Requested server ID. If the value of **status** is smaller than **300**, the value of this field is **None**.
resource	str	Error source (a bucket or an object). If the value of **status** is smaller than **300**, the value of this field is **None**.
header	list	Response header list, composed of tuples. Each tuple consists of two elements, respectively corresponding to the key and value of a response header.
body	object	Result content returned after the operation is successful. If the value of **status** is larger than **300**, the value of this field is **None**. The value varies with the API. For details, see the *OBS Python SDK API Reference*.

19.4 Log Analysis

Log Configuration

OBS Python SDK provides the logging function based on the Python log library. You can call **ObsClient.initLog** to enable and configure logging. Sample code is as follows:

```
# Import the module.
from obs import LogConf
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)

# Specify the path to the log configuration file and initialize logs of ObsClient.
obsClient.initLog(LogConf('./log.conf'), 'obs_logger');
```

NOTE

- The logging function is disabled by default. You need to enable it if needed.
- The log configuration file example (**log.conf**) is included in the OBS Python SDK development package. Modify parameters in **log.conf** as needed.

NOTICE

The log module of the OBS Python SDK is thread safe but not process safe. If ObsClient is used in multi-process scenarios, you must configure an independent log path for each instance of ObsClient to prevent conflicts when multiple processes write logs concurrently.

Log Format

The SDK log format is: *Log time*|*Process ID*|*Thread number*|*Log level*|*Log content*. The following are example logs:

```
2017-11-06 13:46:54,936|process:6100|thread:12700|DEBUG|HTTP(s)+XML|OBS_LOGGER|__parse_xml,188|
http response result:status:200,reason:OK,code:None,message:None,headers:[('id-2',
'LgOKocHfuHe0rFSUHS6LcChzcoYes0luPgqxhUfCP58xp3MZh2n4YKRPpABV8GEK'), ('connection', 'close'),
('request-id', '0001AFF8E60000015F8FDA1EA5AE04E3'), ('date', 'Mon, 06 Nov 2017 05:42:37 GMT'),
('content-type', 'application/xml')]|
2017-11-06 13:46:54,937|process:6100|thread:12700|INFO|HTTP(s)+XML|OBS_LOGGER|doClose,349|server
inform to close connection|
2017-11-06 13:46:54,937|process:6100|thread:12700|INFO|HTTP(s)+XML|OBS_LOGGER|wrapper,59|
listBuckets cost 56 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 **DEBUG** logs and the least information in **ERROR** logs.

Log level description:

- **DEBUG**: Debugging level. If this level is set, all logs will be printed.
- **INFO**: Information level. If this level is set, logs at the **WARNING** level and the time consumed for each HTTP/HTTPS request will be printed.
- **WARNING**: Warning level. If this level is set, logs at the **ERROR** level and some critical events will be printed.
- **ERROR**: Error level. If this level is set, only error information will be printed.

NOTE

In the configuration file, **LogFileLevel** is used to specify the log level for log files, and **PrintLogLevel** is used to specify the log level for the console.

20 FAQ

20.1 How Can I Create a Folder?

To create a folder in an OBS bucket is to create an object whose size is 0 and whose name ends with a slash (/). For details, see **Creating a Folder**.

20.2 How Can I List All Objects in a Bucket?

For details, see **Listing Objects** and **Listing Versioning Objects**.

20.3 How Can I Use a URL for Authorized Access?

See **10.1 Using a URL for Authorized Access**.

20.4 How Can I Upload an Object in Browser-Based Mode?

For details, see **Performing a Browser-Based Upload**.

20.5 How Can I Download a Large Object in Multipart Mode?

For details, see **Performing a Partial Download**.

20.6 What Can I Do to Implement Server-Side Root Certificate Verification?

For details, see **Configuring Server-Side Certificate Verification**.

20.7 How Can I Set an Object to Be Accessible to Anonymous Users?

To enable anonymous users to access an object, perform the following steps:

Step 1 Set the object access permission to **public-read** by referring to **9.2 Managing Object ACLs**.

Step 2 Obtain the URL of the object by referring to **20.11 How Do I Obtain the Object URL?** and provide it to anonymous users.

Step 3 An anonymous user can access the object by entering the URL on a browser.

----End

20.8 How Can I Identify the Endpoint and Region of OBS?

For details, see **Obtaining Endpoints**.

20.9 How Can I Use pip to Download the SDK?

See **Installing the SDK**.

20.10 What Is the Retry Mechanism of SDK?

SDK uses **max_retry_count** configured in **4.4 Configuring an Instance of ObsClient** to retry. The default value for retry times is **3**. A value ranges from **0** to **5** is recommended. If the network connection is abnormal or the server returns the 5*XX* error when an ObsClient API is called, the SDK performs an exponential backoff retry.

NOTICE

- When **ObsClient.putContent** is called for a streaming upload, the SDK does not retry when an I/O exception occurs because the data stream cannot be read back. The upper-layer application needs to retry.
- When **ObsClient.getObject** is successfully called and the object that supports the streaming download is returned, the SDK does not retry when an I/O error occurs during data reading from the returned object because this situation is beyond the scope of the processing logic of the SDK. The upper-layer application needs to retry.

20.11 How Do I Obtain the Object URL?

If the uploaded object is set to be read by anonymous users, anonymous users can download the object through the object URL directly. Methods to obtain the object URL are as follows:

Method 1: Query by calling the API. After an object is uploaded by calling **ObsClient.putContent** or **ObsClient.putFile**, **PutContentResponse** is returned. You can call **objectUrl** to obtain the URL of the uploaded object. Sample code is as follows:

```
# Import the module.
from obs import ObsClient

# Create an instance of ObsClient.
obsClient = ObsClient(
    access_key_id='*** Provide your Access Key ***',
    secret_access_key='*** Provide your Secret Key ***',
    server='https://your-endpoint'
)
resp = obsClient.putContent('bucketname', 'objectname', content='Hello OBS')
if resp.status < 300:
    print('requestId:', resp.requestId)
    print('objectUrl:', resp.body.objectUrl)
else:
    print('requestId:', resp.requestId)
    print('errorCode:', resp.errorCode)
```

Method 2: Compose the URL in the format of **https://***Bucket name.Domain name*/*Directory level*/*Object name*.

NOTE

- If the object resides in the root directory of a bucket, its URL does not contain a directory level.
- You can click **here** to view the domain names of each region.
- For example, if you want to access the object named **test.txt** in the **test** folder of bucket **testbucket** residing in region CN North-Beijing4, the URL of the object is https://testbucket.obs.cn-north-4.myhuaweicloud.com/test/test.txt.

20.12 How to Improve the Speed of Uploading Large Files over the Public Network?

If the size of a file exceeds 100 MB, you are advised to upload the file using multipart upload over the public network. Multipart upload allows uploading a single object as parts separately. Each part is a part of consecutive object data. You can upload parts in any sequence. A part can be reloaded after an upload failure, without affecting other parts. Uploading multiple parts of an object using multiple threads concurrently can greatly improve the transmission efficiency.

For details about the code example, see **7.8 Performing a Multipart Upload**.

20.13 How Can I Perform a Multipart Upload?

For details, see **7.8 Performing a Multipart Upload**.

20.14 How Can I Perform a Download in Multipart Mode?

In a multipart download, you can specify the range of data to be downloaded. The procedure is as follows:

Step 1 You need to initialize an instance of ObsClient by using AK, SK, and endpoint.

Step 2 Call **GetObjectHeader.range** to set the start and end points of the object data to be downloaded.

Step 3 Call **ObsClient.getObject** to send the request to obtain the object in step 2 to download the data in multipart mode.

----End

For details, see **8.5 Performing a Partial Download**.

20.15 How Can I Obtain the AK and SK?

Step 1 Log in to OBS Console. In the upper right corner of the page, hover the cursor over the username and click **My Credentials**.

Step 2 On the **My Credentials** page, select **Access Keys** in the navigation pane on the left.

Step 3 On the **Access Keys** page, click **Create Access Key**.

Step 4 In the **Create Access Key** dialog box that is displayed, enter the password and verification code.

Step 5 Click **OK**.

Step 6 In the **Download Access Key** dialog box that is displayed, click **OK** to save the access keys to your browser's default download path.

Step 7 Open the downloaded **credentials.csv** file to obtain the access keys (AK and SK).

----End

A API Reference

For details about all parameters and definitions of APIs in the OBS Python SDK, see the **Object Storage Service Python SDK API Reference**.

B Change History

Release Date	What's New
2019-11-20	This is the thirteenth official release. Reorganized the positions of the following sections: • Example Programs • Technical Support Channels
2019-09-05	This is the twelfth official release. Modified the following sections: Added methods of creating an instance of ObsClient in **4.2 Creating an Instance of ObsClient**. Added ObsClient configuration parameters in **4.4 Configuring an Instance of ObsClient**.
2019-08-15	This is the eleventh official release. Added the following sections: **5 Fault Locating** **19.1 HTTP Status Codes** **20 FAQ**
2019-05-30	This is the tenth official release. Added the following section: • **A API Reference** Modified the following sections: • Modified the SDK source code address in **1 SDK Download Links**. • In **3.2 Setting Up an OBS Environment**, added the content related to the temporary access keys.

Release Date	What's New
2019-03-05	This is the ninth official release. Modified the following section: • Added the description and code example for the function workflow service in each section of "Event Notification." • Modified the code example in section **14.2 Enabling Bucket Logging**.
2019-01-17	This is the eighth official release. Modified the following section: Added the SDK source code address in **1 SDK Download Links**. Added the composer installation method in **3.4 Installing the SDK**. Optimized the description in **10.1 Using a URL for Authorized Access**. Added rules for combining object link address formats in **7.1 Object Upload Overview**.
2018-10-31	This is the seventh official release. Modified the following content: • Added section "Obtaining Upload Progress." • Added section "Obtaining Download Progress."
2018-08-31	This is the sixth official release. Modified the following content: • Added the **is_cname** parameter in section "Configuring an Instance of ObsClient."
2018-07-18	This is the fifth official release. Modified the following content: • Added section "Managing Bucket Policies." • Added section "Performing an Appendable Upload." • Added section "Configuring Lifecycle Management."
2018-06-30	This is the fourth official release. Modified the following content: • Resolved some known issues.
2018-01-31	This is the third official release. Modified the following content: • Added "Example Programs." • Modified sample code in "Temporary Authorized Request."

Release Date	What's New
2017-12-31	This is the second official release. Modified the following content: ● Added "Processing an Image." ● Added the object transition policy feature in "Lifecycle Management."
2017-11-30	This is the first official release.

