Object Storage Service

PHP SDK Developer Guide (Ally Region)

Issue	01
Date	2026-01-15

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property 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 Cloud 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.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 SDK Download Links

Download Address

- Latest version of OBS PHP SDK: **Download**

Compatibility

- Recommended PHP versions: 5.6 and 7.*x*

> **CAUTION**
>
> - PHP SDK 3.22.6 and later require the PHP version must not be earlier than PHP 7.1.
> - PHP SDK 3.23.11 and earlier are compatible with up to PHP 8.1. To be compatible with later PHP versions, upgrade the PHP SDK version to 3.24.9.

- Namespace: This version is incompatible with the earlier version (2.1.*x*). All public classes and functions are saved in the **obs** package.
- Interface functions: Not completely compatible with earlier versions (2.1.*x*). The following table describes the changes.

Interface Function	Description
ObsClient.setBucketCors	In the request parameters, the **CorsRule** field is renamed as **CorsRules**.
ObsClient.getBucketCors	In the response parameters, the **CorsRule** field is renamed as **CorsRules**.
ObsClient.setBucketTagging	In the request parameters, the **TagSet** field is renamed as **Tags**.
ObsClient.getBucketTagging	In the response parameters, the **TagSet** field is renamed as **Tags**.

2 Example Programs

OBS PHP SDK provides abundant example programs for your reference and use.

These programs can be obtained from the OBS PHP SDK.

After you decompress **eSDK_Storage_OBS_*<VersionId>*_PHP.zip**, you can obtain example programs from **eSDK_Storage_OBS_*<VersionId>*_PHP/src/examples**.

3 Quick Start

3.1 Before You Start

- You can see **General Examples of ObsClient** to understand how to call OBS PHP SDK APIs in a general manner.
- After an API of **ObsClient** is called, if **SDK common result objects** are returned with no exception thrown, the operation is successful. If there is an exception returned, the operation fails, and you can reference **SDK custom exceptions** to learn the reason.
- Some features are available only in some regions. If an API call returns the 405 HTTP status code, check whether the region supports this feature.

3.2 Preparations

Before using OBS SDK for PHP to access OBS, you need to prepare the service and development environments. To prepare the service environment, you will need an account and access keys. Both of them are necessary for interaction between OBS SDK and OBS. To ensure successful SDK installation and SDK-based code development and running, you should also set up a local development environment, for example, installing dependencies and development tools.

Preparing Access Keys

Access keys consist of two parts: an access key ID (AK) and a secret access key (SK). OBS uses access keys to sign requests to make sure that only authorized accounts can access specified OBS resources. Programmatic access must be enabled for an IAM user before the IAM user can get access keys. Access keys are explained as follows:

- One AK maps to only one user but one user can have multiple AKs. OBS authenticates users by their AKs.
- An SK is required for accessing OBS. Authentication information is generated based on the SK and request headers. AKs and SKs are in one-to-one match.

Access keys are classified into permanent access keys (AK/SK) and temporary access keys (AK/SK and security token). Each user can create at most two

permanent access keys. Temporary access keys must be used within a given validity period. Once expired, they must be requested again. For security purposes, you are advised to use temporary access keys to access OBS. If you want to use permanent access keys, periodically update them. The following describes how to obtain two types of access keys.

- To get permanent access keys, do as follows:
  a. Log in to the management console.
  b. In the upper right corner, hover over the username and choose **My Credentials**.
  c. On the **My Credentials** page, click **Access Keys** in the navigation pane.
  d. On the **Access Keys** page, click **Create Access Key**.
  e. In the displayed dialog box, enter the login password and verification code.

     NOTE

     - If you have not bound an email address or a mobile number yet, only the login password is required.
     - If you have bound both an email address and a mobile number, you can use either of them for verification.

  f. Click **OK**.
  g. Click **Download**. The access key file is automatically saved to your browser's default download path.
  h. Open the downloaded **credentials.csv** file to obtain the access keys (AK and SK).

     NOTE

     - Each user can create a maximum of two valid access key pairs.
     - Keep AKs and SKs properly to prevent information leakage. If you click **Cancel** in the download dialog box, the access keys will not be downloaded and cannot be downloaded later. You can create a new AK/SK pair if needed.
- To get temporary access keys, refer to the following:

  Temporary access keys are issued by the system and are only valid for 15 minutes to 24 hours. Once expired, they must be requested again. They follow the principle of least privilege. When a temporary AK/SK pair is used for authentication, a security token must be used at the same time.

Setting Up a Development Environment

- Download a recommended version from the **PHP official website** and install it.
- (Optional) Download the latest version of PhpStorm from the **JetBrains official website** and install it.

  NOTE

  After PHP is installed, you need to set the extension library parameter (**extension_dir**) in the **php.ini** file and start the cURL and OpenSSL extension libraries.

3.3 Downloading and Installing the SDK

Table 3-1 shows you how to install the PHP SDK.

Table 3-1 Installation methods

No.	Method
1	Manually downloading and installing the source code development package

Manually Downloading and Installing the Source Code Development Package

The following uses OBS PHP SDK of the latest version as an example:

Step 1 Download the OBS PHP SDK development package by referring to **Downloading the SDK**.

Step 2 Decompress the development package to obtain the following files: **examples** (the sample code), **Obs** (the SDK source code), **composer.json** (the dependency configuration file), **obs-autoloader.php** (the file for automatically loading PHP dependency libraries), and **README.txt** (the feature description file of SDK versions).

Step 3 In the CLI, go to the directory where the SDK is decompressed and run the **composer install** command to install the dependencies. A folder named **vendor** will be generated.

Step 4 (Optional) In the PhpStorm project, import the source code. Open PhpStorm, choose **File** > **Open**, and select the directory where the SDK is decompressed in **Open File or Project**.

----End

NOTE

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

```
├── examples
├── Obs
├── vendor
├── composer.json
├── obs-autoloader.php
└── README.txt
```

3.4 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:


```
// Before initializing an instance of ObsClient, you must import dependencies.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';

// Declare the namespace.
use Obs\ObsClient;

// Create an ObsClient instance.
$obsClient = new ObsClient([
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
]);

// Use the instance to access OBS.

// Close ObsClient.
$obsClient -> close();
```

NOTE

For more information, see chapter "Initialization."

For details about logging configuration, see **Configuring SDK Logging**.

3.5 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:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;

// Create an ObsClient instance.
$obsClient = new ObsClient([
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
]);

$resp = $obsClient -> createBucket([
     'Bucket' => 'bucketname',
     // Set the ACL for the bucket to public read (the default state is private read and write).
     'ACL' => ObsClient::AclPublicRead,
     // Set the bucket storage class to Standard.
     'StorageClass' => ObsClient::StorageClassStandard,
     // Set the bucket location.
     'LocationConstraint' => 'bucketlocation'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
```

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**.
- For more information, see **Creating a Bucket**.

3.6 Uploading an Object

This example uploads string **Hello OBS** to bucket **bucketname** as object **objectname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;

// Create an instance of ObsClient.
$obsClient = new ObsClient([
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
]);

$resp = $obsClient -> putObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'Body' => 'Hello OBS'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
```

NOTE

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

3.7 Downloading an Object

This example downloads object **objectname** from bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
```



```
// Declare the namespace.
use Obs\ObsClient;

// Create an instance of ObsClient.
$obsClient = new ObsClient([
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
]);

$resp = $obsClient -> getObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
echo $resp ['Body'];
```

NOTE

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

3.8 Listing Objects

This example lists objects in bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;

// Create an instance of ObsClient.
$obsClient = new ObsClient([
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
]);

$resp = $obsClient -> listObjects([
     'Bucket' => 'bucketname'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
foreach ( $resp ['Contents'] as $index => $content ) {
     printf("Contents[$index][Key]:%s\n", $content ['Key']);
     printf("Contents[$index][LastModified]:%s\n", $content ['LastModified']);
     printf("Contents[$index][Size]:%s\n", $content ['Size']);
}
```

NOTE

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

3.9 Deleting an Object

This example deletes object **objectname** from bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;

// Create an instance of ObsClient.
$obsClient = new ObsClient([
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
]);

$resp = $obsClient -> deleteObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
```

NOTE

- This example only deletes a single object. To delete objects in a batch, traverse objects and list to-be-deleted objects on your own.
- For details about deletion, see **Deleting Objects**.

3.10 General Examples of ObsClient

Each time you call an API of **ObsClient**, you need to pass the associative array to the request as the input. For a bucket-related API, the **Bucket** field contained in the associative array is used to specify the bucket name (excluding **ObsClient->listBuckets**). For an object-related API, the **Bucket** field and **Key** field contained in the associative array are used to specify the bucket name and object name, respectively. **ObsClient** supports synchronous and asynchronous API callings. Examples are as follows:

Synchronous Method Call

If any exception is thrown when an API is called by using the synchronous method, the operation fails. Otherwise, the operation succeeds.

Sample code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
```

```
$obsClient = new ObsClient([
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
]);

// Construct bucket request parameters.
$requestParam1 = [
     'Bucket' => 'bucketname'
     // Other fields.
];

try{
     // Use the synchronous method to call a bucket-related API, such as the API for creating a bucket.
     $resp = $obsClient->createBucket ( $requestParam1 );
     // If the operation is successful, handle the API calling result.
     printf ( "RequestId:%s\n", $resp ['RequestId'] );
}catch (Obs\ObsException $obsException){
     // If the operation fails, obtain the exception details.
     printf("ExceptionCode:%s\n", $obsException->getExceptionCode());
     printf("ExceptionMessage:%s\n", $obsException->getExceptionMessage());
}

// Construct object request parameters.
$requestParam2 = [
     'Bucket' => 'bucketname',
     'Key' => 'objectname'
     // Other fields.
];

try{
     // Use the synchronous method to call an object-related API, such as the API for downloading an object.
     $resp = $obsClient->getObject ( $requestParam2 );
     // If the operation is successful, handle the API calling result.
     printf ( "RequestId:%s\n", $resp ['RequestId'] );
}catch (Obs\ObsException $obsException){
     // If the operation fails, obtain the exception details.
     printf("ExceptionCode:%s\n", $obsException->getExceptionCode());
     printf("ExceptionMessage:%s\n", $obsException->getExceptionMessage());
}

// Close obsClient.
$obsClient -> close();
```

Asynchronous Method Call

In the asynchronous method call mode, the calling result is returned by the callback function. If the SDK custom exception is not null, the operation fails. Otherwise, the operation succeeds. Sample code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient([
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
```

```
]);

// Construct bucket request parameters.
$requestParam1 = [
    'Bucket' => 'bucketname'
    // Other fields.
];

// Use the asynchronous method to call a bucket-related API, such as the API for creating a bucket.
$promise1 = $obsClient->createBucketAsync ( $requestParam1 , function($obsException, $resp){
    if($obsException !== null){
        // If the operation fails, obtain the exception details.
        printf("ExceptionCode:%s\n", $obsException->getExceptionCode());
        printf("ExceptionMessage:%s\n", $obsException->getExceptionMessage());
    }else{
        // If the operation is successful, handle the API calling result.
        printf ( "RequestId:%s\n", $resp ['RequestId'] );
    }
});

// Wait for the result of calling bucket-related APIs.
$promise1 -> wait();

// Construct object request parameters.
$requestParam2 = [
    'Bucket' => 'bucketname',
    'Key' => 'objectname'
    // Other fields.
];

// Use the asynchronous method to call an object-related API, such as the API for downloading an object.
$promise2 = $obsClient->getObjectAsync ( $requestParam2 , function($obsException, $resp){
    if($obsException !== null){
        // If the operation fails, obtain the exception details.
        printf("ExceptionCode:%s\n", $obsException->getExceptionCode());
        printf("ExceptionMessage:%s\n", $obsException->getExceptionMessage());
    }else{
        // If the operation is successful, handle the API calling result.
        printf ( "RequestId:%s\n", $resp ['RequestId'] );
    }
});

// Wait for the result of calling object-related APIs.
$promise2 -> wait();

// Close obsClient.
$obsClient -> close();
```

3.11 Pre-defined Constants

OBS PHP SDK provides a group of pre-defined constants that can be directly used.

You can call **ObsClient** to obtain the pre-defined constants.

For details about predefined constants, see the *OBS PHP SDK API Reference*.

4 Initialization

4.1 Creating and Configuring an OBS Client

Scenarios

This section describes how to create and configure an OBS client.

Prerequisites

Ensure you have completed the following preparations:

1. Select a proper SDK version by referring to **Before You Start**.
2. Prepare the service and development environments by referring to **Preparations**.
3. Download and install the OBS SDK for PHP by referring to **Downloading and Installing the SDK**.

Configuring Parameters

Table 4-1

Parameter	Description	Recommended Value
key	AK	N/A
secret	SK	N/A
endpoint	Endpoint for accessing OBS, which contains the protocol type, domain name (or IP address), and port name. For example, **https://***your-endpoint***:443**. For security purposes, you are advised to use HTTPS.	N/A

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 default CAs are used to verify the server-side certificate. • **false**: The server-side certificates will not be verified. The default value is **false**.	N/A
max_retry_count	Maximum number of retries when an HTTP/HTTPS connection is abnormal. The default value is **3**.	[1, 5]
socket_timeout	Timeout duration for transmitting data at the socket layer, in seconds. The default value is **60**.	[10, 60]
connect_timeout	Timeout period for establishing an HTTP/HTTPS connection, in seconds. The default value is **60**.	[10, 60]
chunk_size	Block size for reading socket streams, in bytes. The default value is **65536**.	Default value
is_cname	Whether to use a user-defined domain name to access OBS. The default value is **false**. **NOTE** To protect your services from being affected by the takeover of Huawei Cloud public domain names by relevant organizations, you are advised to use a user-defined domain name to access a bucket.	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 larger values for **socket_timeout** and **connect_timeout**.
- If the **endpoint** you specified does not contain a protocol, HTTPS is used by default.

By Using the Constructor

```
// Declare the namespace.
use Obs\ObsClient;

// Create an instance of ObsClient.
```

```
$obsClient = new ObsClient([
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    // (Optional) If you use a temporary AK/SK pair and a security token to access OBS, you are not advised to use hard coding, which may result in information leakage. You can obtain an AK/SK pair using environment variables or import an AK/SK pair in other ways.
    // 'security_token' => getenv('SECURITY_TOKEN'),
    'endpoint' => 'https://your-endpoint',
]);

// Use the instance to access OBS.

// Close obsClient.
$obsClient -> close();
```

By Using the Factory Method

```
// Declare the namespace.
use Obs\ObsClient;

// Create an instance of ObsClient.
$obsClient = ObsClient::factory ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    // (Optional) If you use a temporary AK/SK pair and a security token to access OBS, you are not advised to use hard coding, which may result in information leakage. You can obtain an AK/SK pair using environment variables or import an AK/SK pair in other ways.
    // 'security_token' => getenv('SECURITY_TOKEN'),
    'endpoint' => 'https://your-endpoint',
]);

// Use the instance to access OBS.

// Close obsClient.
$obsClient -> close();
```

NOTE

- The project can contain one or more instances of **ObsClient**.
- After you call the **ObsClient -> close** method to close an instance of **ObsClient**, the instance cannot be used any more.

4.2 Configuring SDK Logging

OBS PHP SDK provides the logging function based on the monolog log library. You can call **ObsClient->initLog** to enable and configure logging. Sample code is as follows:

```
$obsClient -> initLog ([
    'FilePath' => './logs', // Set the log folder.
    'FileName' => 'eSDK-OBS-PHP.log', // Set the name for the log file.
    'MaxFiles' => 10, // Set the maximum number of log files that can be retained.
    'Level' => WARN // Set the log level.
]);
```


NOTE

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

4.3 Asynchronous Method Call

All bucket- and object-related APIs provided by OBS PHP SDK can be called by asynchronous methods whose names end with **Async** (such as **ObsClient->putObjectAsync** if the synchronous method is named **ObsClient->putObject**). The returned result will be output to a callback function. A callback function contains an **SDK custom exception** and an **SDK common result object** in sequence. If the **SDK common result object** is not null, the operation fails. Otherwise, the operation succeeds.

The following code shows how to upload an object in asynchronous method call mode:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

// Upload the object in asynchronous method call mode.
$promise = $obsClient->putObjectAsync ( [
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'Body' => 'Hello OBS'
], function ($obsException, $resp) {
    if ($obsException === null) {
        printf ( "RequestId:%s\n", $resp ['RequestId'] );
    } else {
        printf ( "ExceptionCode:%s\n", $obsException->getExceptionCode () );
        printf ( "ExceptionMessage:%s\n", $obsException->getExceptionMessage () );
    }
} );
$promise->wait ();
```

NOTE

A result object (**GuzzleHttp\Promise\Promise**) will be returned upon an asynchronous method call. You need to call the **wait** method of the object to wait until the asynchronous method call is complete.

5 Bucket Management

5.1 Creating a Bucket

You can call **ObsClient->createBucket** to create a bucket. Sample code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

// Create a bucket.
$resp = $obsClient->createBucket([
     'Bucket' => 'bucketname',
     // Set the ACL for the bucket to public read (the default state is private).
     'ACL' => ObsClient::AclPublicRead,
     // Set the bucket storage class to Standard.
     'StorageClass' => ObsClient::StorageClassStandard,
     // Set the bucket location.
     'LocationConstraint' => 'bucketlocation'
]);
printf("RequestId:%s\n", $resp['RequestId']);
```

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 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 example is of the default ACL (private), in the OBS Standard storage class, and in the default region.
- You can use parameter **ACL** to specify the bucket ACL, use **StorageClass** to specify the storage class of the bucket, and use **LocationConstraint** to specify the bucket location.

5.2 Listing Buckets

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

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.     'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->listBuckets([
     'QueryLocation' => true
]);

printf("RequestId:%s\n", $resp['RequestId']);
printf("Owner[ID]:%s\n", $resp['Owner']['ID']);
foreach ($resp['Buckets'] as $index => $bucket){
     printf("Buckets[%d]\n", $index + 1);
     printf("Name:%s\n", $bucket['Name']);
     printf("CreationDate:%s\n", $bucket['CreationDate']);
     printf("Location:%s\n", $bucket['Location']);
}
```

NOTE

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

5.3 Deleting a Bucket

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

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.     'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

// Delete a bucket.
$resp = $obsClient->deleteBucket([
      'Bucket' => 'bucketname'
]);

printf("RequestId:%s\n", $resp['RequestId']);
```

NOTE

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

5.4 Identifying Whether a Bucket Exists

You can call **ObsClient->headBucket** to identify whether a bucket exists.

This example checks whether bucket **bucketname** exists.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.     'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

try{
    $resp = $obsClient->headBucket([
        'Bucket' => 'bucketname'
    ]);
    printf("Bucket exists");
}catch (\Obs\Common\ObsException $obsException){
    if($obsException->getStatusCode() === 404){
```

```
        printf("Bucket does not exist");
    }else{
        printf("ExceptionCode:%s\n", $obsException->getExceptionCode());
        printf("getExceptionMessage:%s\n", $obsException->getExceptionMessage());
    }
}
```

NOTE

- If an exception is thrown and the returned HTTP status code is **404**, the bucket does not exist.

5.5 Obtaining Bucket Metadata

You can call **ObsClient->getBucketMetadata** to obtain the metadata of a bucket.

This example returns the metadata of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketMetadata([
     'Bucket' => 'bucketname',
     'Origin' => 'http://www.a.com'
]);

printf("RequestId:%s\n",$resp['RequestId']);
printf("StorageClass:%s\n",$resp['StorageClass']);
printf("AllowOrigin:%s\n",$resp['AllowOrigin']);
printf("MaxAgeSeconds:%s\n",$resp['MaxAgeSeconds']);
printf("ExposeHeader:%s\n",$resp['ExposeHeader']);
printf("AllowMethod:%s\n",$resp['AllowMethod']);
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

5.6 Managing Bucket ACLs

Access control lists (ACLs) allow resource owners to grant other accounts the permissions to access resources. By default, only the resource owner has full control over resources when a bucket or object is created. That is, the bucket creator has full control over the bucket, and the object uploader has full control over the object. Other accounts do not have the permissions to access resources. If resource owners want to grant other accounts the read and write permissions on resources, they can use ACLs. ACLs grant permissions to accounts. After an

account is granted permissions, both the account and its IAM users can access the resources.

1. Specify a pre-defined ACL when creating a bucket.
2. Call ObsClient->setBucketAcl to specify a pre-defined ACL.
3. Call ObsClient->setBucketAcl to specify a user-defined ACL.

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

Permission	Description	Value in OBS PHP 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.	ObsClient::PermissionRead
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.	ObsClient::PermissionWrite
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.	ObsClient::PermissionReadAcp
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.	ObsClient::PermissionWriteAcp
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**, **READ_ACP**, and **WRITE_ACP** permissions for the object.	ObsClient::PermissionFullControl

The following table lists the five types of pre-defined ACLs of OBS:

Policy	Description	Value in OBS PHP 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.	ObsClient::AclPrivate
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.	ObsClient::AclPublicRead
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.	ObsClient::AclPublicReadWrite
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.	ObsClient::AclPublicReadDelivered

Policy	Description	Value in OBS PHP SDK
public-read-write-delivered	If this permission is set for a bucket, everyone can obtain the object list in the bucket, multipart tasks 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.	ObsClient::AclPublicReadWriteDelivered

Specifying a Pre-defined ACL During Bucket Creation

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.     'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

// Create a bucket.
$resp = $obsClient->createBucket([
     'Bucket' => 'bucketname',
     // Set the bucket ACL to public read and write.
     'ACL' => ObsClient::AclPublicReadWrite
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

Setting a Pre-defined ACL for a Bucket

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.     'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
```



```
    'endpoint' => 'https://your-endpoint'
] );

// Set a pre-defined ACL.
$resp = $obsClient->setBucketAcl([
      'Bucket' => 'bucketname',
      // Set the bucket ACL to private read and write.
      'ACL' => ObsClient::AclPrivate
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

Setting a User-defined Bucket ACL

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

// Set a user-defined bucket ACL.
$resp = $obsClient->setBucketAcl([
      'Bucket' => 'bucketname',
      // Set the bucket owner.
      'Owner' => [
            'ID' => 'ownerid'
      ],
      'Grants' => [
            // Grant all permissions to a specified user.
            ['Grantee' => ['Type' => 'CanonicalUser', 'ID' => 'userid'], 'Permission' => ObsClient::PermissionFullControl],
            // Grant the READ permission to all users.
            ['Grantee' => ['Type' => 'Group', 'URI' => ObsClient::GroupAllUsers], 'Permission' => ObsClient::PermissionRead],
      ]
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

NOTE

- Use the **Owner** parameter to specify the bucket owner and the **Grants** parameter to specify the information about authorized users.
- The owner or grantee ID required in the ACL indicates an account ID, which can be viewed on the **My Credentials** page of OBS Console.
- OBS buckets support the following grantee group:
  - All users: ObsClient::GroupAllUsers

Obtaining a Bucket ACL

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

```
// Import the dependency library.
require 'vendor/autoload.php';
```



```
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketAcl([
     'Bucket' => 'bucketname'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
printf ("Owner[ID]:%s\n", $resp ['Owner']['ID']);
foreach ( $resp ['Grants'] as $index => $grant ) {
     printf ("Grants[%d]\n", $index + 1);
     printf ("Grantee[ID]:%s\n", $grant['Grantee']['ID']);
     printf ("Grantee[URI]:%s\n", $grant['Grantee']['URI']);
     printf ("Permission:%s\n", $grant['Permission']);
}
```

5.7 Managing Bucket Policies

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

Setting a Bucket Policy

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

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$bucketName = "bucketname";
$policy = "{\"Statement\":[{\"Principal\":\"*\",\"Effect\":\"Allow\",\"Action\":\"ListBucket\",\"Resource\":\"" .$bucketName. "\"}]}";

$resp = $obsClient->setBucketPolicy([
     'Bucket' => $bucketName,
     'Policy' => $policy
]);

printf("RequestId:%s\n",$resp['RequestId']);
```


NOTE

For details about the bucket policies in JSON format, see *OBS PHP SDK API Reference*.

Obtaining a Bucket Policy

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

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketPolicy([
     'Bucket' => 'bucketname'
]);

printf("RequestId:%s\n",$resp['RequestId']);
printf("Policy:%s\n",$resp['Policy']);
```

Deleting a Bucket Policy

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

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->deleteBucketPolicy([
     'Bucket' => 'bucketname'
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

5.8 Obtaining a Bucket Location

You can call **ObsClient->getBucketLocation** to obtain the location of a bucket.

This example returns the region of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketLocation([
     'Bucket' => 'bucketname'
]);

printf("RequestId:%s\n",$resp['RequestId']);
printf("Location:%s\n",$resp['Location']);
```

NOTE

- When creating a bucket, you can specify its location. For details, see **Creating a Bucket**.
- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

5.9 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.

This example returns the storage information of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketStorageInfo([
     'Bucket' => 'bucketname'
]);

printf("RequestId:%s\n",$resp['RequestId']);
printf("Size:%s\n",$resp['Size']);
printf("ObjectNumber:%s\n",$resp['ObjectNumber']);
```


NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

5.10 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->setBucketQuota([
     'Bucket' => 'bucketname',
     'StorageQuota' => 1024 * 1024 * 100
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

NOTE

- Use the **StorageQuota** parameter to specify the bucket quota.
- 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketQuota([
     'Bucket' => 'bucketname'
]);
```



```
printf("RequestId:%s\n",$resp['RequestId']);
printf("StorageQuota:%s\n",$resp['StorageQuota']);
```

5.11 Storage Class

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 PHP 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.	ObsClient::StorageClassStandard
		ObsClient::StorageClassWarm
		ObsClient::StorageClassCold
Intelligent Tiering	Is designed to optimize storage costs by automatically moving data to a more economical access tier when data access patterns change. This storage class is ideal for data with constantly changing or unpredictable access patterns.	ObsClient::StorageClassINTELLIGENT_TIERING

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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->setBucketStoragePolicy([
```



```
    'Bucket' => 'bucketname',
    'StorageClass' => ObsClient::StorageClassWarm
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

NOTE

Use the **StorageClass** parameter to set the storage class for a bucket.

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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketStoragePolicy([
     'Bucket' => 'bucketname'
]);

printf("RequestId:%s\n",$resp['RequestId']);
printf("StorageClass:%s\n",$resp['StorageClass']);
```

6 Object Upload

6.1 Object Upload Overview

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

- **Performing a Text-Based Upload**
- **Performing a Streaming Upload**
- **Performing a File-Based Upload**
- **Performing a Multipart Upload**
- **Performing a Browser-Based Upload**

The SDK supports the upload of objects whose size ranges from 0 KB to 5 GB. If a file is smaller than 5 GB, streaming upload and file-based upload are applicable. If the file is larger than 5 GB, multipart upload (whose part size is smaller than 5 GB) is suitable. Browser-based upload supports the file to be uploaded through a browser.

If you grant anonymous users the read permission for an object during the upload, anonymous users can access the object through a URL after the upload is complete. 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.

6.2 Performing a Text-Based Upload

Text-based upload is used to directly upload character strings. You can call **ObsClient->putObject** to upload character strings to OBS.

This example uploads string **Hello OBS** to bucket **bucketname** as object **objectname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
```



```
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'Body' => 'Hello OBS'
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

NOTE

Use the **Body** parameter to specify the character string to be uploaded.

6.3 Performing a Streaming Upload

Streaming upload uses **resource** or **GuzzleHttp\Psr7\StreamInterface** as the data source of an object. Sample code is as follows:

Uploading a Network Stream

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     // Create network streams.
     'Body' => fopen('http://www.a.com','r')
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

Uploading a File Stream

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
```

```
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'Body' => fopen('localfile', 'r')
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

NOTICE

- If the **Body** parameter is used to specify the to-be-uploaded streaming data, its value must be a **resource** or **GuzzleHttp\Psr7\StreamInterface** object.
- To upload a large file, you are advised to use **multipart upload**.

6.4 Performing a File-Based Upload

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

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'SourceFile' => 'localfile'  // Path of the local file to be uploaded. The file name must be specified.
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

NOTE

- Use the **SourceFile** parameter to specify the path to the to-be-uploaded file.
- **SourceFile** and **Body** cannot be used together.
- The content to be uploaded cannot exceed 5 GB.

6.5 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
     'Key' => 'parent_directory/'
]);

printf("RequestId:%s\n",$resp['RequestId']);
// Create an object in the folder.
$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
     'Key' => 'parent_directory/objectname',
     'Body' => 'Hello OBS',
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

NOTE

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

6.6 Setting Object Properties

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 to 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

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'SourceFile' => 'localfile',   // Path of the local file to be uploaded. The file name must be specified.
     'ContentLength' => 1024 * 1024 // 1 MB
]);
```



```
printf("RequestId:%s\n",$resp['RequestId']);
```

NOTE

Use the **ContentLength** parameter to specify the object length.

Setting the MIME Type for an Object

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname.jpg',
     'SourceFile' => 'localimage.jpg',
     'ContentType' => 'image/jpeg'
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

NOTE

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

Setting the MD5 Value for an Object

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'Body' => 'Hello OBS'
     // your md5 which should be encoded by base64
     'ContentMD5' => base64_encode(hash("md5", "Hello OBS!", true))
]);

printf("RequestId:%s\n",$resp['RequestId']);
```


NOTE

- Use the **ContentMD5** parameter to specify the MD5 value for an object.
- The MD5 value of an object must be a base64-encoded digest.
- The OBS server will compare this MD5 value with the MD5 value obtained by object data calculation. If the two values are not the same, the upload fails with an HTTP **400** error returned.
- If the MD5 value is not specified, the OBS server will skip MD5 value verification.

Setting the Storage Class for an Object

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'SourceFile' => 'localfile',
     'StorageClass' => ObsClient::StorageClassCold
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

NOTE

- Use the **StorageClass** parameter to set the storage class for an object.
- 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.

Customizing Metadata for an Object

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->putObject([
     'Bucket' => 'bucketname',
```



```
    'Key' => 'objectname',
    'SourceFile' => 'localfile',
        'Metadata' => ['property1' => 'property-value1', 'property2' => 'property-value2']
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

NOTE

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

6.7 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

Initiating a Multipart Upload

Before using a multipart upload, you need to first let OBS initiate it. 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
```

```
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->initiateMultipartUpload([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'ContentType' => 'text/plain',
     'Metadata' => ['property' => 'property-value']
]);

printf("RequestId:%s\n",$resp['RequestId']);
printf("UploadId:%s\n",$resp['UploadId']);
```

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. The last part uploaded ranges from 0 to 5 GB in size, and **each of the other parts ranges 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->uploadPart([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     // Set the part number, which ranges from 1 to 10000.
     'PartNumber' => 1,
     // Set the upload ID.
     'UploadId' => 'upload id from initiateMultipartUpload',
// Set the large file to be uploaded. localfile is the path of the local file to be uploaded. You need to specify
```



```
the file name.
    'SourceFile' => 'localfile',
    // Set the part size.
    'PartSize' => 5 * 1024 * 1024,
    // Set the start offset.
    'Offset' => 0
]);

printf("RequestId:%s\n",$resp['RequestId']);
printf("ETag:%s\n",$resp['ETag']);
```

NOTE

- Use the **PartNumber** parameter to specify the part number, the **UploadId** parameter to specify the globally unique ID, the **SourceFile** parameter to specify the to-be-uploaded file, the **PartSize** parameter to set the part size, and the **Offset** parameter to set the start offset of a part.
- 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.
- You can use the **ContentMD5** parameter to set the MD5 value of the uploaded data.
- 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->completeMultipartUpload([
    'Bucket' => 'bucketname',
    'Key' => 'objectname',
    // Set the upload ID.
    'UploadId' => 'upload id from initiateMultipartUpload',
    'Parts' => [
            ['PartNumber' => 1, 'ETag' => 'etag value from uploadPart']
    ]
]);
```



```
printf("RequestId:%s\n",$resp['RequestId']);
```

CAUTION

- If the size of a part other than the last part is smaller than 100 KB, OBS returns **400 Bad Request**.

NOTE

- Use the **UploadId** parameter to specify the globally unique identifier for the multipart upload and the **Parts** parameter to specify the list of part numbers and ETags. Content in the list is displayed in the ascending order by part number.
- Part numbers can be inconsecutive.

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 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->abortMultipartUpload([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     // Set the upload ID.
     'UploadId' => 'upload id from initiateMultipartUpload'
]);

printf("RequestId:%s\n",$resp['RequestId']);
```

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 **ObsClient->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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->listParts ( [
    'Bucket' => 'bucketname',
    'Key' => 'objectname',
    'UploadId' => 'upload id from initiateMultipartUpload'
] );

printf ( "RequestId:%s\n", $resp ['RequestId'] );
foreach ( $resp ['Parts'] as $index => $part ) {
    printf ( "Parts[%d]\n", $index + 1 );
     // Part number, specified upon uploading
    printf ( "PartNumber:%s\n", $part ['PartNumber'] );
     // Time when the part was last uploaded
    printf ( "LastModified:%s\n", $part ['LastModified'] );
     // Part ETag
    printf ( "ETag:%s\n", $part ['ETag'] );
    // Part size
    printf ( "Size:%s\n", $part ['Size'] );
}
```

NOTE

- A maximum of 1,000 parts can be listed each time. If the upload of a specified ID contains more than 1,000 parts, **IsTruncated** in the response is **true**, indicating not all parts were listed. In such case, you can use **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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$partNumberMarker = null;
$index = 1;
do{
    $resp = $obsClient->listParts ( [
        'Bucket' => 'bucketname',
        'Key' => 'objectname',
        'UploadId' => 'upload id from initiateMultipartUpload',
        'PartNumberMarker' => $partNumberMarker
    ] );

    printf ( "RequestId:%s\n", $resp ['RequestId'] );
    foreach ( $resp ['Parts'] as $part ) {
        printf ( "Parts[%d]\n", $index );
        // Part number, specified upon uploading
        printf ( "PartNumber:%s\n", $part ['PartNumber'] );
        // Time when the part was last uploaded
        printf ( "LastModified:%s\n", $part ['LastModified'] );
        // Part ETag
        printf ( "ETag:%s\n", $part ['ETag'] );
        // Part size
        printf ( "Size:%s\n", $part ['Size'] );
        $index ++;
    }

    $partNumberMarker = $resp['NextPartNumberMarker'];

}while($resp['IsTruncated']);
```

Listing Multipart Uploads

You can call **ObsClient->listMultipartUploads** to list multipart uploads. The following table describes related parameters.

Parameter	Description
Prefix	Prefix that the object names in the multipart uploads to be listed must contain
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.)

Parameter	Description
MaxUploads	Maximum number of listed multipart uploads. The value ranges from 1 to 1000. If the value is not in this range, 1000 parts are listed by default.
KeyMarker	Object name to start with when listing multipart uploads
UploadIdMarker	Upload ID after which the multipart upload listing begins. It is effective only when used with **KeyMarker** so that multipart uploads after **UploadIdMarker** of **KeyMarker** will be listed.

- Listing multipart uploads in simple mode

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->listMultipartUploads ( [
    'Bucket' => 'bucketname'
] );

printf ( "RequestId:%s\n", $resp ['RequestId'] );
foreach ( $resp ['Uploads'] as $index => $upload ) {
    printf ( "Uploads[%d]\n", $index + 1 );
    printf ( "Key:%s\n", $upload ['Key'] );
    printf ( "UploadId:%s\n", $upload ['UploadId'] );
    printf ( "Initiated:%s\n", $upload ['Initiated'] );
    printf ( "Owner[ID]:%s\n", $upload ['Owner'] ['ID'] );
    printf ( "StorageClass:%s\n", $upload ['StorageClass'] );
}
```

NOTE

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

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
```



```
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$keyMarker = null;
$uploadIdMarker = null;
$index = 1;
do{
    $resp = $obsClient->listMultipartUploads ( [
        'Bucket' => 'bucketname',
        'KeyMarker' => $keyMarker,
        'UploadIdMarker' => $uploadIdMarker
    ] );

    printf ( "RequestId:%s\n", $resp ['RequestId'] );
    foreach ( $resp ['Uploads'] as $index => $upload ) {
        printf ( "Uploads[%d]\n", $index );
        printf ( "Key:%s\n", $upload ['Key'] );
        printf ( "UploadId:%s\n", $upload ['UploadId'] );
        printf ( "Initiated:%s\n", $upload ['Initiated'] );
        printf ( "Owner[ID]:%s\n", $upload ['Owner'] ['ID'] );
        printf ( "StorageClass:%s\n", $upload ['StorageClass'] );
        $index ++;
    }
    $keyMarker = $resp['NextKeyMarker'];
    $uploadIdMarker = $resp['NextUploadIdMarker'];
}while ($resp['IsTruncated']);
```

6.8 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.

This example copies object **sourceobjectname** from bucket **sourcebucketname** to bucket **destbucketname** as object **destobjectname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$destBucketName = 'destbucketname';
$destObjectKey = 'destobjectname';
$sourceBucketName = 'sourcebucketname';
$sourceObjectKey = 'sourceobjectname';

// Initiate a multipart upload.
$resp = $obsClient->initiateMultipartUpload ( [
```



```
    'Bucket' => $destBucketName,
    'Key' => $destObjectKey
] );
$uploadId = $resp ['UploadId'];
printf ( "UploadId:%s\n\n", $uploadId );

// Obtain information about the large object.
$resp = $obsClient->getObjectMetadata ( [
    'Bucket' => $sourceBucketName,
    'Key' => $sourceObjectKey
]);

// Set the part size to 100 MB.
$partSize = 100 * 1024 * 1024;

$objectSize = $resp ['ContentLength'];

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

// Start copying parts concurrently.
$promise = null;
$parts = [];
for($i = 0; $i < $partCount; $i ++) {
    $rangeStart = $i * $partSize;
    $rangeEnd = ($i + 1 === $partCount) ? $objectSize - 1 : $rangeStart + $partSize - 1;
    $partNumber = $i + 1;
    $p = $obsClient->copyPartAsync ( [
        'Bucket' => $destBucketName,
        'Key' => $destObjectKey,
        'UploadId' => $uploadId,
        'PartNumber' => $partNumber,
        'CopySource' => sprintf ( '%s/%s', $sourceBucketName, $sourceObjectKey ),
        'CopySourceRange' => sprintf ( 'bytes=%d-%d', $rangeStart, $rangeEnd )
    ], function ($exception, $resp) use (&$parts, $partNumber) {
        $parts [] = [
            'PartNumber' => $partNumber,
            'ETag' => $resp ['ETag']
        ];
        printf ( "Part#" . strval ( $partNumber ) . " done\n\n" );
    } );

    if ($promise === null) {
        $promise = $p;
    }
}

// Wait until the copy is complete.
$promise->wait ();

usort ( $parts, function ($a, $b) {
    if ($a ['PartNumber'] === $b ['PartNumber']) {
        return 0;
    }
    return $a ['PartNumber'] > $b ['PartNumber'] ? 1 : - 1;
} );


// Combine parts.
$resp = $obsClient->completeMultipartUpload ( [
    'Bucket' => $destBucketName,
    'Key' => $destObjectKey,
    'UploadId' => $uploadId,
    'Parts' => $parts
] );

printf("Complete to upload multiparts finished, RequestId:%s\n", $resp['RequestId']);
```

NOTE

Use the **PartNumber** parameter to specify the part number, the **UploadId** parameter to specify the globally unique ID for the multipart upload, the **CopySource** parameter to specify the information about the source object, and the **CopySourceRange** parameter to specify the copy range.

6.9 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.

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**: which 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->createPostSignature([
     // Set the validity period for the browser-based upload request, in seconds.
     'Expires' => 3600,
     // Fill in parameters in the form.
     'FormParams' => [
          // Set the object ACL to public read.
          'x-obs-acl' => ObsClient::AclPublicRead,
          // Set the MIME type for the object.
          'content-type' => 'text/plain',
     ]
```



```
]);

// Obtain the request parameters.
printf("Policy:%s\n", $resp['Policy']);
printf("Signature:%s\n", $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
<!-- Object ACL -->
<input type="text" name="x-obs-acl" value="public-read" />
<p>
Content-Type
<!-- Object MIME type -->
<input type="text" name="content-type" value="text/plain" />
<p>
<!-- Base64 code of the policy -->
<input type="hidden" name="policy" value="*** Provide your policy ***" />
<!-- AK -->
<input type="hidden" name="AccessKeyId" value="*** Provide your access key ***"/>
<!-- Signature information -->
<input type="hidden" name="signature" value="*** Provide your signature ***"/>

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

NOTE

- Values of **policy** and **signature** in the HTML form are obtained from the value returned by **ObsClient.createPostSignatureSync**.

7 Object Download

7.1 Object Download Overview

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

- **Performing a Text-Based Download**
- **Performing a Streaming Download**
- **Performing a File-Based Download**
- **Performing a Partial Download**
- **Performing a Conditioned Download**

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

7.2 Performing a Text-Based Download

This example downloads object **objectname** from bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient -> getObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname'
```



```
]);

printf("RequestId:%s\n", $resp['RequestId']);
printf("Object Content:\n");
// Obtain the object content.
echo $resp ['Body'];
```

NOTE

- When this download mode is adopted, **Body** in the returned result is an instance of **GuzzleHttp\Psr7\StreamInterface**, which contains text content.
- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

7.3 Performing a Streaming Download

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient -> getObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'SaveAsStream' => true
]);

printf("RequestId:%s\n", $resp['RequestId']);
printf("Object Content:\n");
while(!$resp['Body'] -> eof()){
     echo $resp['Body'] -> read(65536);
}
```

NOTE

- Use the **SaveAsStream** parameter to specify the download mode to streaming download.
- **Body** in the returned result is a readable **GuzzleHttp\Psr7\StreamInterface** object and can be used to save the object to a local directory or to the memory.

7.4 Performing a File-Based Download

This example downloads object **objectname** from bucket **bucketname** as **localfile**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
```



```
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient -> getObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'SaveAsFile' => 'localfile',
]);

printf("RequestId:%s\n", $resp['RequestId']);
```

NOTE

- Use the **SaveAsFile** parameter to specify the path for saving the to-be-downloaded file.
- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

7.5 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 from 0 to 1,000, data from byte 0 to byte 1,000, 1,001 bytes in total, are returned. If the specified range is invalid, data of the whole object will be returned. Sample code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient -> getObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'Range' => 'bytes=0-1000'
]);

printf("RequestId:%s\n", $resp['RequestId']);
printf("Object Content:\n");
echo $resp ['Body'];
```


NOTE

- Use the **Range** parameter to specify the download range in the format of "bytes=*x-y*."
- 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.

7.6 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
IfModifiedSince	Returns the object if it has been modified since the specified time; otherwise, an error is returned.	This parameter must conform to the HTTP time format specified in http://www.ietf.org/rfc/rfc2616.txt.
IfUnmodifiedSince	Returns the object if it has not been modified since the specified time; otherwise, an error is returned.	This parameter must conform to the HTTP time format specified in http://www.ietf.org/rfc/rfc2616.txt.
IfMatch	Returns the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.	Character string
IfNoneMatch	Returns the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.	Character string

NOTE

- The ETag of an object is the MD5 check value of the object.
- If a request includes **IfUnmodifiedSince** or **IfMatch** and the specified condition is not met, the object download will fail and an exception will be thrown with error code **412 Precondition Failed** returned.
- If a request includes **IfModifiedSince** or **IfNoneMatch** and the specified condition is not met, the object download will fail and an exception will be thrown with error code **304 Not Modified** returned.

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient -> getObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'IfModifiedSince' => 'Thu, 31 Dec 2015 16:00:00 GMT'
]);

printf("RequestId:%s\n", $resp['RequestId']);
printf("Object Content:\n");
echo $resp ['Body'];
```

7.7 Rewriting Response Headers

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

Parameter	Description
ResponseContentType	Rewrites **Content-Type** in HTTP/HTTPS responses.
ResponseContentLanguage	Rewrites **Content-Language** in HTTP/HTTPS responses.
ResponseExpires	Rewrites **Expires** in HTTP/HTTPS responses.
ResponseCacheControl	Rewrites **Cache-Control** in HTTP/HTTPS responses.
ResponseContentDisposition	Rewrites **Content-Disposition** in HTTP/HTTPS responses.

Parameter	Description
ResponseContentEncoding	Rewrites **Content-Encoding** in HTTP/HTTPS responses.

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient -> getObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'ResponseContentType' => 'image/jpeg'
]);

printf("RequestId:%s\n", $resp['RequestId']);
// Obtain the rewritten response headers.
printf("ContentType:%s\n", $resp['ContentType']);
```

7.8 Obtaining Customized Metadata

After an object is successfully downloaded, its customized data is returned.

This example downloads object **objectname** from **bucketname** and returns the custom metadata of the object.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient -> getObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname'
```



```
]);

printf("RequestId:%s\n", $resp['RequestId']);
printf("Metadata:%s\n", print_r($resp['Metadata'], true));
```

NOTE

- If **Metadata** is left blank, the object has no custom metadata configured.
- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

7.9 Downloading a Cold Object

Before you can download a Cold object, you must restore it. Cold objects can be restored in either of the following ways.

Option	Description	Value on the OBS Server
Expedited restore	Data can be restored within 1 to 5 minutes.	ObsClient::RestoreTierExpedited
Standard restore	Data can be restored within 3 to 5 hours. This is the default option.	ObsClient::RestoreTierStandard

CAUTION

To prolong the validity period of the Cold data restored, you can repeatedly restore the data, but you will be billed for each restoration. After a second restore, the validity period of Standard object copies will be prolonged, and you need to pay for storing these copies during the prolonged period.

You can call **ObsClient->restoreObject** to restore a Cold object. Sample code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

// Restore a Cold object.
$resp = $obsClient -> restoreObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'Days' => 1,
```

```
    'Tier' => ObsClient::RestoreTierExpedited
]);

printf("RequestId:%s\n", $resp['RequestId']);

// Wait for the object to be restored.
sleep(6 * 60);

$resp = $obsClient -> getObject([
    'Bucket' => 'bucketname',
    'Key' => 'objectname'
]);

printf("RequestId:%s\n", $resp['RequestId']);
printf("Object Content:\n");
// Obtain the object content.
echo $resp ['Body'];
```

NOTE

- The object specified in **ObsClient->restoreObject** must be in the OBS Cold storage class. Otherwise, an error will be reported when you call this API.
- Use the **Days** parameter to specify the retention period (from 1 to 30 days) of the restored objects and the **Tier** parameter to specify the time spent on restoring the objects.

8 Object Management

8.1 Obtaining Object Properties

You can call **ObsClient->getObjectMetadata** to obtain properties of an object, including the length, MIME type, customized metadata.

This example obtains the metadata of **objectname** in **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient([
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
]);

$resp = $obsClient -> getObjectMetadata([
     'Bucket' => 'bucketname',
     'Key' => 'objectname'
]);

printf("RequestId:%s\n", $resp['RequestId']);
printf("ContentType:%s\n", $resp['ContentType']);
printf("ContentLength:%s\n", $resp['ContentLength']);
printf("Metadata:%s\n", print_r($resp['Metadata'], true));
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

8.2 Managing Object ACLs

Access control lists (ACLs) allow resource owners to grant other accounts the permissions to access resources. By default, only the resource owner has full control over resources when a bucket or object is created. That is, the bucket creator has full control over the bucket, and the object uploader has full control over the object. Other accounts do not have the permissions to access resources. If resource owners want to grant other accounts the read and write permissions on resources, they can use ACLs. ACLs grant permissions to accounts. After an account is granted permissions, both the account and its IAM users can access the resources.

1. Specify a pre-defined ACL during object upload.
2. Call ObsClient->setObjectAcl to specify the pre-defined ACL.
3. Call ObsClient->setObjectAcl to specify a user-defined ACL.

Specifying a Pre-defined ACL During Object Upload

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient -> putObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'Body' => 'Hello OBS',
     // Set the object ACL to public read.
     'ACL' => ObsClient::AclPublicRead
]);

printf("RequestId:%s\n", $resp['RequestId']);
```

Setting a Pre-defined ACL for an Object

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
```



```
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient -> setObjectAcl([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     // Set the object to be private.
     'ACL' => ObsClient::AclPrivate
]);

printf("RequestId:%s\n", $resp['RequestId']);
```

Setting an Object ACL Directly

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient -> setObjectAcl([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     // Set the object owner.
     'Owner' => ['ID' => 'ownerid'],
     'Grants' => [
          // Grant all permissions to a specified user.
          ['Grantee' => ['Type' => 'CanonicalUser', 'ID' => 'userid'], 'Permission' => ObsClient::PermissionFullControl],
          // Grant the READ permission to all users.
          ['Grantee' => ['Type' => 'Group', 'URI' => ObsClient::AllUsers], 'Permission' => ObsClient::PermissionRead],
    ]
]);

printf("RequestId:%s\n", $resp['RequestId']);
```

NOTE

- Use the **Owner** parameter to specify the object owner and the **Grants** parameter to specify information about the authorized users.
- The owner or grantee ID required in the ACL indicates an account ID, which can be viewed on the **My Credentials** page of OBS Console.
- OBS buckets support the following grantee group:
  - All users: ObsClient::GroupAllUsers

Obtaining an Object ACL

You can call ObsClient->getObjectAcl to obtain the ACL of an object. Sample code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->getObjectAcl ( [
     'Bucket' => 'bucketname',
     'Key' => 'objectname'
] );

printf ( "RequestId:%s\n", $resp ['RequestId'] );
printf ( "Owner[ID]:%s\n", $resp ['Owner'] ['ID'] );
foreach ( $resp ['Grants'] as $index => $grant ) {
     printf ( "Grants[%d]\n", $index + 1 );
     printf ( "Grantee[ID]:%s\n", $grant ['Grantee'] ['ID'] );
     printf ( "Grantee[URI]:%s\n", $grant ['Grantee'] ['URI'] );
     printf ( "Permission:%s\n", $grant ['Permission'] );
}
```

8.3 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.
MaxKeys	Maximum number of objects listed in the response. The value ranges from 1 to 1000. If the value is not in this range, 1000 objects are listed by default.

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.) For a parallel file system, if this parameter is not specified, all the content in the directory is recursively listed by default, and subdirectories are also listed. In big data scenarios, parallel file systems usually have deep directory levels and each directory has a large number of files. In such case, you are advised to configure **[delimiter='/']** to list the content in the current directory, but not list subdirectories, thereby improving the listing efficiency.

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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->listObjects ( [
     'Bucket' => 'bucketname'
] );

printf ( "RequestId:%s\n", $resp ['RequestId'] );
foreach ( $resp ['Contents'] as $index => $content ) {
     printf ( "Contents[%d]\n", $index + 1 );
     printf ( "Key:%s\n", $content ['Key'] );
     printf ( "LastModified:%s\n", $content ['LastModified'] );
     printf ( "ETag:%s\n", $content ['ETag'] );
     printf ( "Size:%s\n", $content ['Size'] );
     printf ( "Owner[ID]:%s\n", $content ['Owner'] ['ID'] );
     printf ( "StorageClass:%s\n", $content ['StorageClass'] );
}
```

NOTE

- A maximum of 1,000 objects can be listed each time. If a bucket contains more than 1,000 objects, **IsTruncated** in the response is **true**, indicating not all objects were listed. In such case, you can use **NextMarker** 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->listObjects ( [
     'Bucket' => 'bucketname',
     // Specify the number of objects to be listed to 100.
     'MaxKeys' => 100
] );

printf ( "RequestId:%s\n", $resp ['RequestId'] );
foreach ( $resp ['Contents'] as $index => $content ) {
     printf ( "Contents[%d]\n", $index + 1 );
     printf ( "Key:%s\n", $content ['Key'] );
     printf ( "LastModified:%s\n", $content ['LastModified'] );
     printf ( "ETag:%s\n", $content ['ETag'] );
     printf ( "Size:%s\n", $content ['Size'] );
     printf ( "Owner[ID]:%s\n", $content ['Owner'] ['ID'] );
     printf ( "StorageClass:%s\n", $content ['StorageClass'] );
}
```

Listing Objects by Specifying a Prefix

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->listObjects ( [
     'Bucket' => 'bucketname',
     // Set the prefix to prefix and the number of objects to be listed to 100.
     'MaxKeys' => 100,
     'Prefix' => 'prefix'
] );

printf ( "RequestId:%s\n", $resp ['RequestId'] );
foreach ( $resp ['Contents'] as $index => $content ) {
     printf ( "Contents[%d]\n", $index + 1 );
     printf ( "Key:%s\n", $content ['Key'] );
```



```
    printf ( "LastModified:%s\n", $content ['LastModified'] );
    printf ( "ETag:%s\n", $content ['ETag'] );
    printf ( "Size:%s\n", $content ['Size'] );
    printf ( "Owner[ID]:%s\n", $content ['Owner'] ['ID'] );
    printf ( "StorageClass:%s\n", $content ['StorageClass'] );
}
```

Listing Objects by Specifying the Start Position

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.     'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->listObjects ( [
    'Bucket' => 'bucketname',
    // Set that 100 objects whose names follow test in lexicographical order will be listed.
    'MaxKeys' => 100,
    'Marker' => 'test'
] );

printf ( "RequestId:%s\n", $resp ['RequestId'] );
foreach ( $resp ['Contents'] as $index => $content ) {
    printf ( "Contents[%d]\n", $index + 1 );
    printf ( "Key:%s\n", $content ['Key'] );
    printf ( "LastModified:%s\n", $content ['LastModified'] );
    printf ( "ETag:%s\n", $content ['ETag'] );
    printf ( "Size:%s\n", $content ['Size'] );
    printf ( "Owner[ID]:%s\n", $content ['Owner'] ['ID'] );
    printf ( "StorageClass:%s\n", $content ['StorageClass'] );
}
```

Listing All Objects in Paging Mode

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$marker = null;
$index = 1;
```

```
do {

    $resp = $obsClient->listObjects ( [
        'Bucket' => 'bucketname',
        // Set the number of parts displayed per page to 100.
        'MaxKeys' => 100,
        'Marker' => $marker
    ] );

    printf ( "RequestId:%s\n", $resp ['RequestId'] );
    foreach ( $resp ['Contents'] as $content ) {
        printf ( "Contents[%d]\n", $index );
        printf ( "Key:%s\n", $content ['Key'] );
        printf ( "LastModified:%s\n", $content ['LastModified'] );
        printf ( "ETag:%s\n", $content ['ETag'] );
        printf ( "Size:%s\n", $content ['Size'] );
        printf ( "Owner[ID]:%s\n", $content ['Owner'] ['ID'] );
        printf ( "StorageClass:%s\n", $content ['StorageClass'] );
        $index ++;
    }
    $marker = $resp['NextMarker'];
}while($resp['IsTruncated']);
```

Listing All Objects in a Folder

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

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$marker = null;
$index = 1;
do {

    $resp = $obsClient->listObjects ( [
        'Bucket' => 'bucketname',
        'MaxKeys' => 1000,
        // Set the prefix of the folders to dir/.
        'Prefix' => 'dir/',
        'Marker' => $marker
    ] );

    printf ( "RequestId:%s\n", $resp ['RequestId'] );
    foreach ( $resp ['Contents'] as $content ) {
        printf ( "Contents[%d]\n", $index );
        printf ( "Key:%s\n", $content ['Key'] );
        printf ( "LastModified:%s\n", $content ['LastModified'] );
        printf ( "ETag:%s\n", $content ['ETag'] );
        printf ( "Size:%s\n", $content ['Size'] );
        printf ( "Owner[ID]:%s\n", $content ['Owner'] ['ID'] );
        printf ( "StorageClass:%s\n", $content ['StorageClass'] );
```



```
            $index ++;
        }
        $marker = $resp['NextMarker'];
}while($resp['IsTruncated']);
```

Listing All Objects According to Folders in a Bucket

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

function listObjectsByPrefix($commonPrefiexes){
    global $obsClient;
    foreach ($commonPrefiexes as $commonPrefiex){
        $resp = $obsClient->listObjects ( [
                'Bucket' => 'bucketname',
                // Set folder isolators to slashes (/).
                'Delimiter' => '/',
                'Prefix' => $commonPrefiex['Prefix']
        ] );
        printf("Objects in folder [%s]:\n", $commonPrefiex['Prefix']);
        foreach ( $resp ['Contents'] as $index => $content ) {
            printf ( "Contents[%d]\n", $index );
            printf ( "Key:%s\n", $content ['Key'] );
            printf ( "LastModified:%s\n", $content ['LastModified'] );
            printf ( "ETag:%s\n", $content ['ETag'] );
            printf ( "Size:%s\n", $content ['Size'] );
            printf ( "Owner[ID]:%s\n", $content ['Owner'] ['ID'] );
            printf ( "StorageClass:%s\n", $content ['StorageClass'] );
        }
        printf("\n");
        listObjectsByPrefix($resp['CommonPrefixes']);
    }
}

$resp = $obsClient->listObjects ( [
    'Bucket' => 'bucketname',
    // Set folder isolators to slashes (/).
    'Delimiter' => '/'
] );

printf( "Objects in the root directory:\n");
foreach ( $resp ['Contents'] as $index => $content ) {
    printf ( "Contents[%d]\n", $index );
    printf ( "Key:%s\n", $content ['Key'] );
    printf ( "LastModified:%s\n", $content ['LastModified'] );
    printf ( "ETag:%s\n", $content ['ETag'] );
    printf ( "Size:%s\n", $content ['Size'] );
    printf ( "Owner[ID]:%s\n", $content ['Owner'] ['ID'] );
    printf ( "StorageClass:%s\n", $content ['StorageClass'] );
}
printf("\n");
listObjectsByPrefix($resp['CommonPrefixes']);
```


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, **Contents** includes the objects in the folder and **CommonPrefixes** includes the sub-folders in the folder.

8.4 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->deleteObject ( [
     'Bucket' => 'bucketname',
     'Key' => 'objectname'
] );
printf("RequestId:%s\n", $resp['RequestId']);
```

Batch Deleting Objects

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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->deleteObjects ( [
            'Bucket' => 'bucketname',
            // Set the response mode to verbose.
            'Quiet' => false,
            'Objects' => [
                        [
                                    'Key' => 'objectname1',
                                    'VersionId' => null
                        ],
                        [
                                    'Key' => 'objectname2',
                                    'VersionId' => null
                        ]
            ]
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
// Obtain the successfully deleted objects.
printf ( "Deleteds:\n" );
foreach ( $resp ['Deleteds'] as $index => $deleted ) {
    printf ( "Deleteds[%d]", $index + 1 );
    printf ( "Key:%s\n", $deleted ['Key'] );
    printf ( "VersionId:%s\n", $deleted ['VersionId'] );
    printf ( "DeleteMarker:%s\n", $deleted ['DeleteMarker'] );
    printf ( "DeleteMarkerVersionId:%s\n", $deleted ['DeleteMarkerVersionId'] );
}
// Obtain the objects failed to be deleted.
printf ( "Errors:\n" );
foreach ( $resp ['Errors'] as $index => $error ) {
    printf ( "Errors[%d]", $index + 1 );
    printf ( "Key:%s\n", $error ['Key'] );
    printf ( "VersionId:%s\n", $$error ['VersionId'] );
    printf ( "Code:%s\n", $error ['Code'] );
    printf ( "Message:%s\n", $error ['Message'] );
}
```

NOTE

Use the **Quiet** parameter to specify the response mode and the **Objects** parameter to specify the to-be-deleted objects.

8.5 Copying an Object

Function

This API copies an object stored in OBS to another path. You can copy an object of up to 5 GB in a single operation.

You can use this API to create a copy for an object in a specified bucket.

Restrictions

- To copy an object, you must be the bucket owner or have the required permission (**obs:object:PutObject** in IAM or **PutObject** in a bucket policy).
- You must have the read permission for the source object.
- The object copy request contains the information about the source bucket and object to be copied in the header field. The message body cannot be contained.
- Cross-bucket replication in the same region is supported, but cross-region replication is not supported.
- The target object size ranges from 0 to 5 GB. If the source object size exceeds 5 GB, you must use a multipart copying API by referring to **Performing a Multipart Copy**.
- If the source object is in the Cold storage class, you must restore it first.

Method

```
ObsClient->copyObject(array $parameter)
```

Request Parameters

Table 8-1 List of request parameters

Parameter	Type	Mandatory (Yes/No)	Description
Bucket	string	Yes	**Definition:** Destination bucket name. **Restrictions:** ● A bucket name must be unique across all accounts and regions. ● A bucket name: – Must be 3 to 63 characters long and start with a digit or letter. Lowercase letters, digits, hyphens (-), and periods (.) are allowed. – Cannot be formatted as an IP address. – Cannot start or end with a hyphen (-) or period (.). – Cannot contain two consecutive periods (..), for example, **my..bucket**. – Cannot contain a period (.) and a hyphen (-) adjacent to each other, for example, **my-.bucket** or **my.-bucket**. ● If you repeatedly create buckets with the same name in the same region, no error will be reported and the bucket attributes comply with those set in the first creation request. **Default value:** None

Parameter	Type	Mandatory (Yes/No)	Description
Key	string	Yes	**Definition:** Target object name. An object is uniquely identified by an object name in a bucket. An object name is a complete path of the object that does not contain the bucket name. **Value range:** The value must contain 1 to 1,024 characters. **Default value:** None
CopySource	string	Yes	**Definition:** Parameter used to specify the source bucket, source object, and source object version ID (optional). **Restrictions:** Format: *Source bucket name*/*Source object name*?**versionId=***Source object version ID* **Default value:** None
ACL	string	No	**Definition:** ACL specified for the object. You can use either a pre-defined or a user-defined ACL. **Value range:** To use a pre-defined ACL, see **Table 8-2** for the available options. **Default value:** AccessControlList.REST_CANNED_PRIVATE
StorageClass	string	No	**Definition:** Object storage class. **Value range:** See **Table 8-3**. **Default value:** None. If this parameter is not specified, the storage class of the bucket is used as that of the object.

Parameter	Type	Mandatory (Yes/No)	Description
CopySourceIfMatch	string	No	**Definition:** If the ETag of the source object is the same as the one specified by this parameter, it is copied. Otherwise, an error is returned. **Default value:** None
CopySourceIfNoneMatch	string	No	**Definition:** If the ETag of the source object is different from the one specified by this parameter, it is copied. Otherwise, an error is returned. **Default value:** None
CopySourceIfUnmodifiedSince	string or \DateTime	No	**Definition:** If the source object has not been modified since the specified time, it is copied. Otherwise, an exception is thrown. **Default value:** None
CopySourceIfModifiedSince	string or \DateTime	No	**Definition:** If the source object has been modified since the specified time, it is copied. Otherwise, an exception is thrown. **Default value:** None
CacheControl	string	No	**Definition:** Rewrites the **Cache-Control** header in the response. **Default value:** None
ContentDisposition	string	No	**Definition:** Rewrites the **Content-Disposition** header in the response. **Default value:** None

Parameter	Type	Mandatory (Yes/No)	Description
ContentEncoding	string	No	**Definition:** Rewrites the **Content-Encoding** header in the response. **Default value:** None
ContentLanguage	string	No	**Definition:** Rewrites the **Content-Language** header in the response. **Default value:** None
ContentType	string	No	**Definition:** Rewrites the **Content-Type** header in the response. **Default value:** None
Expires	string	No	**Definition:** Rewrites the **Expires** header in the response. **Default value:** None
MetadataDirective	string	No	**Definition:** Policy for copying the source object's attributes. **Value range:** See **Table 8-4**. **Default value:** None

Parameter	Type	Mandatory (Yes/No)	Description
Metadata	associative array	No	**Definition:** Custom metadata of the target object. You can add a header starting with **x-obs-meta-** in the request to define metadata. The custom metadata will be returned in the response when you retrieve the object or query the object metadata. **Restrictions:** ● The custom metadata cannot exceed 8 KB in total. To measure the size, calculate the sum of bytes of all UTF-8 encoded keys and values. ● The custom metadata keys are case insensitive, but are stored in lowercase in OBS. The key values are case sensitive. ● Both custom metadata keys and their values must conform to US-ASCII standards. If non-ASCII or unrecognizable characters are required, they must be encoded and decoded in URL or Base64 on the client, because the server does not perform such operations. **Default value:** None

Parameter	Type	Mandatory (Yes/No)	Description
WebsiteRedirectLocation	string	No	**Definition:** If the bucket is configured with website hosting, the request for obtaining the object can be redirected to another object in the bucket or an external URL. To another object in the same bucket: **WebsiteRedirectLocation:/anotherPage.html** To an external URL: **WebsiteRedirectLocation:http://www.example.com/** OBS obtains the specified value from the header and stores it in the object metadata **WebsiteRedirectLocation**. **Restrictions:** • The value must start with a slash (/), **http://**, or **https://** and cannot exceed 2 KB. • OBS only supports redirection for objects in the root directory of a bucket. **Default value:** None

Table 8-2 Pre-defined ACL

Access Method	Type	Description
ObsClient::AclPrivate	string	Private read and write.
ObsClient::AclPublicRead	string	Public read.
ObsClient::AclPublicReadWrite	string	Public read and write.
ObsClient::AclPublicReadDelivered	string	Public read on a bucket as well as the objects in the bucket.
ObsClient::AclPublicReadWriteDelivered	string	Public read and write on a bucket as well as the objects in the bucket.

Table 8-3 Storage classes

Access Method	Type	Description
ObsClient::StorageClassStandard	string	Standard storage class.
ObsClient::StorageClassWarm	string	Warm storage class.
ObsClient::StorageClassCold	string	Cold storage class.

Table 8-4 Metadata replication policies

Access Method	Type	Description
ObsClient::CopyMetadata	string	Copies metadata.
ObsClient::ReplaceMetadata	string	Replaces metadata.

Responses

Table 8-5 Responses

Parameter	Type	Description
HttpStatusCode	integer	**Definition:** HTTP status code. **Value range:** A status code is a group of digits that can be **2***xx* (indicating successes) or **4***xx* or **5***xx* (indicating errors). It indicates the status of a response. **Default value:** None
Reason	string	**Definition:** Reason description. **Default value:** None
RequestId	string	**Definition:** Request ID returned by the OBS server. **Default value:** None

Parameter	Type	Description
ETag	string	**Definition:** Base64-encoded, 128-bit MD5 value of an object. ETag is the unique identifier of the object contents and is used to determine whether the contents of an object are changed. For example, if the ETag value is **A** when an object is uploaded and is **B** when the object is downloaded, this indicates the contents of the object are changed. The ETag reflects changes only to the contents of an object, not its metadata. Objects created by the upload and copy operations have unique ETags after being encrypted using MD5. **Restrictions:** If an object is encrypted using server-side encryption, the ETag is not the MD5 value of the object. **Value range:** The value must contain 32 characters. **Default value:** None
LastModified	string	**Definition:** Time when the target object was last modified, in UTC. **Value range:** UTC time **Default value:** None
VersionId	string	**Definition:** Version ID of the target object. **Value range:** The value must contain 32 characters. **Default value:** None

Parameter	Type	Description
CopySourceVersionId	string	**Definition:** Version ID of the source object. **Value range:** The value must contain 32 characters. **Default value:** If versioning is not enabled for the source bucket, this parameter is left blank.

Copying an Object in Simple Mode

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->copyObject ( [
    'Bucket' => 'destbucketname',
    'Key' => 'destobjectname',
    'CopySource' => 'sourcebucketname/sourceobjectname'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

Use the **CopySource** parameter to specify the information about the source object.

Rewriting Object Properties

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
```



```
    'signature' => 'obs'
] );

$resp = $obsClient->copyObject ( [
      'Bucket' => 'destobjectname',
      'Key' => 'destobjectname',
      'CopySource' => 'sourcebucketname/soureobjectname',
      'ContentType' => 'image/jpeg',
      'StorageClass' => ObsClient::StorageClassWarm,
      'Metadata' => ['property' => 'property-value'],
      'MetadataDirective' => ObsClient::ReplaceMetadata
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

Use the **Metadata** parameter to specify the object's customized metadata to be rewritten and the **MetadataDirective** parameter to specify the rewrite mode, which can be **ObsClient::ReplaceMetadata** (rewrite) or **ObsClient::CopyMetadata** (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 exception will be thrown.

You can set the following conditions:

Parameter	Description	Format
CopySourceIfModified-Since	Copies the source object if it has been modified since the specified time; otherwise, an exception is thrown.	This parameter must conform to the HTTP time format specified in http://www.ietf.org/rfc/rfc2616.txt.
CopySourceIfUnmodi-fiedSince	Copies the source object if it has not been modified since the specified time; otherwise, an exception is thrown.	This parameter must conform to the HTTP time format specified in http://www.ietf.org/rfc/rfc2616.txt.
CopySourceIfMatch	Copies the source object if its ETag is the same as the one specified by this parameter; otherwise, an exception is thrown.	Character string
CopySourceIfNoneMatch	Copies the source object if its ETag is different from the one specified by this parameter; otherwise, an exception is thrown.	Character string

NOTE

- The ETag of the source object is the MD5 check value of the source object.
- If the object copy request includes **CopySourceIfUnmodifiedSince**, **CopySourceIfMatch**, **CopySourceIfModifiedSince**, or **CopySourceIfNoneMatch**, and the specified condition is not met, the copy will fail and an exception will be thrown with HTTP status code **412 Precondition Failed** returned.
- **CopySourceIfModifiedSince** and **CopySourceIfNoneMatch** can be used together. So do **CopySourceIfUnmodifiedSince** and **CopySourceIfMatch**.

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->copyObject ( [
     'Bucket' => 'destobjectname',
     'Key' => 'destobjectname',
     'CopySource' => 'sourcebucketname/soureobjectname',
     'CopySourceIfModifiedSince' => 'Thu, 31 Dec 2015 16:00:00 GMT',
     'CopySourceIfNoneMatch' => 'none-match-etag'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

Rewriting an Object ACL

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

$resp = $obsClient->copyObject ( [
     'Bucket' => 'destobjectname',
     'Key' => 'destobjectname',
     'CopySource' => 'sourcebucketname/soureobjectname',
     // Rewrite the object ACL to public read.
     'ACL' => ObsClient::AclPublicRead
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```


NOTE

Use the **ACL** parameter to rewrite the object ACL.

9 Temporarily Authorized Access

9.1 Using a Temporary 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
Configure Bucket ACL	PUT	acl	Yes	No
Obtain 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
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

Operation	HTTP Method	Special Operator (Sub-resource)	Bucket Name Required	Object Name Required
OPTIONS Bucket	OPTIONS	N/A	Yes	No
PUT Object	PUT	N/A	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
Configure Object ACL	PUT	acl	Yes	Yes
Obtain Object ACL	GET	acl	Yes	Yes
Initiate Multipart Upload	POST	uploads	Yes	Yes
PUT Part	PUT	N/A	Yes	Yes
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
OPTIONS Object	OPTIONS	N/A	Yes	Yes
	POST	restore	Yes	Yes

To access OBS using a temporary URL generated by the OBS PHP SDK, perform the following 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 content provides examples of accessing OBS using a temporary URL, including bucket creation, as well as object upload, download, listing, and deletion.

Creating a Bucket

```
// Import the dependency library.
require 'vendor/autoload.php';
```

```
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
use GuzzleHttp\Client;
use GuzzleHttp\Exception\ClientException;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

// Set the validity period of the URL to 3600 seconds.
$expires = 3600;
// Create a bucket.
$resp = $obsClient->createSignedUrl( [
            'Method' => 'PUT',
            'Bucket' => 'bucketname',
            'Expires' => $expires
] );
printf("SignedUrl:%s\n", $resp ['SignedUrl']);

$httpClient = new Client(['verify' => false ]);
$content = '<CreateBucketConfiguration><LocationConstraint>your-location</LocationConstraint></CreateBucketConfiguration>';
$url = $resp['SignedUrl'];
try{
    $response = $httpClient -> request('PUT', $url, ['body' => $content, 'headers'=> $resp['ActualSignedRequestHeaders']]);
    printf("%s using temporary signature url:\n", 'Create bucket');
    printf("\t%s successfully.\n", $url);
    printf("\tStatus:%d\n", $response -> getStatusCode());
    printf("\tContent:%s\n", $response -> getBody() -> getContents());
    $response -> getBody()-> close();
}catch (ClientException $ex){
    printf("%s using temporary signature url:\n", 'Create bucket');
    printf("\t%s failed!\n", $url);
    printf('Exception message:%s', $ex ->getMessage());
}
```

Uploading an Object

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
use GuzzleHttp\Client;
use GuzzleHttp\Exception\ClientException;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

// Set the validity period of the URL to 3600 seconds.
$expires = 3600;
// Upload an object.
```



```
$resp = $obsClient->createSignedUrl( [
            'Method' => 'PUT',
            'Bucket' => 'bucketname',
            'Key' => 'objectname',
            'Expires' => $expires
] );
printf("SignedUrl:%s\n", $resp ['SignedUrl']);
$url = $resp['SignedUrl'];
$httpClient = new Client(['verify' => false ]);
$content = 'Hello OBS';
try{
    $response = $httpClient -> request('PUT', $url, ['body' => $content, 'headers'=>
$resp['ActualSignedRequestHeaders']]);
    printf("%s using temporary signature url:\n", 'Put object');
    printf("\t%s successfully.\n", $url);
    printf("\tStatus:%d\n", $response -> getStatusCode());
    printf("\tContent:%s\n", $response -> getBody() -> getContents());
    $response -> getBody()-> close();
}catch (ClientException $ex){
    printf("%s using temporary signature url:\n", 'Put object');
    printf("\t%s failed!\n", $url);
    printf('Exception message:%s', $ex ->getMessage());
}
```

Downloading an Object

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
use GuzzleHttp\Client;
use GuzzleHttp\Exception\ClientException;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard
coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

// Set the validity period of the URL to 3600 seconds.
$expires = 3600;
// Download an object.
$resp = $obsClient->createSignedUrl( [
            'Method' => 'GET',
            'Bucket' => 'bucketname',
            'Key' => 'objectname',
            'Expires' => $expires
] );
printf("SignedUrl:%s\n", $resp ['SignedUrl']);
$url = $resp['SignedUrl'];
$httpClient = new Client(['verify' => false ]);

try{
    $response = $httpClient -> request('GET', $url, ['headers'=> $resp['ActualSignedRequestHeaders']]);
    printf("%s using temporary signature url:\n", 'Get object');
    printf("\t%s successfully.\n", $url);
    printf("\tStatus:%d\n", $response -> getStatusCode());
    printf("\tContent:%s\n", $response -> getBody() -> getContents());
    $response -> getBody()-> close();
}catch (ClientException $ex){
    printf("%s using temporary signature url:\n", 'Get object');
    printf("\t%s failed!\n", $url);
```

```
    printf('Exception message:%s', $ex ->getMessage());
}
```

Listing Objects

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
use GuzzleHttp\Client;
use GuzzleHttp\Exception\ClientException;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );

// Set the validity period of the URL to 3600 seconds.
$expires = 3600;
// List objects.
$resp = $obsClient->createSignedUrl( [
            'Method' => 'GET',
            'Bucket' => 'bucketname',
            'Expires' => $expires
] );
printf("SignedUrl:%s\n", $resp ['SignedUrl']);
$url = $resp['SignedUrl'];
$httpClient = new Client(['verify' => false ]);

try{
    $response = $httpClient -> request('GET', $url, ['headers'=> $resp['ActualSignedRequestHeaders']]);
    printf("%s using temporary signature url:\n", 'List objects');
    printf("\t%s successfully.\n", $url);
    printf("\tStatus:%d\n", $response -> getStatusCode());
    printf("\tContent:%s\n", $response -> getBody() -> getContents());
    $response -> getBody()-> close();
}catch (ClientException $ex){
    printf("%s using temporary signature url:\n", 'List objects');
    printf("\t%s failed!\n", $url);
    printf('Exception message:%s', $ex ->getMessage());
}
```

Deleting an Object

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
use GuzzleHttp\Client;
use GuzzleHttp\Exception\ClientException;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
    'signature' => 'obs'
] );
```



```
// Set the validity period of the URL to 3600 seconds.
$expires = 3600;
// Delete an object.
$resp = $obsClient->createSignedUrl( [
            'Method' => 'DELETE',
            'Bucket' => 'bucketname',
            'Key' => 'objectname',
            'Expires' => $expires
] );
printf("SignedUrl:%s\n", $resp ['SignedUrl']);
$url = $resp['SignedUrl'];
$httpClient = new Client(['verify' => false ]);

try{
    $response = $httpClient -> request('DELETE', $url, ['headers'=> $resp['ActualSignedRequestHeaders']]);
    printf("%s using temporary signature url:\n", 'Delete object');
    printf("\t%s successfully.\n", $url);
    printf("\tStatus:%d\n", $response -> getStatusCode());
    printf("\tContent:%s\n", $response -> getBody() -> getContents());
    $response -> getBody()-> close();
}catch (ClientException $ex){
    printf("%s using temporary signature url:\n", 'Delete object');
    printf("\t%s failed!\n", $url);
    printf('Exception message:%s', $ex ->getMessage());
}
```

NOTE

Use the **Method** parameter to specify the HTTP request method, the **Expires** parameter to specify the validity period of the URL, the **Headers** parameter to specify the request headers, the **SpecialParam** parameter to specify the special operator, and the **QueryParams** parameter to specify the request parameters.

10 Versioning Management

10.1 Versioning Overview

You can use versioning to store multiple versions of an object in a bucket.

When versioning is enabled for a bucket, OBS keeps multiple versions of an object in the bucket, allowing you to easily retrieve and restore historical versions or recover data in the event of accidental changes or application failures.

By default, versioning is disabled for new OBS buckets. In this case, if a newly uploaded object is using the name of the previously uploaded one, the new object will overwrite the previous one.

For details about different versioning operations, see:

10.2 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	Value on the OBS Server
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.	Enabled
Suspended	1. Noncurrent object versions are not affected. 2. OBS creates version ID **null** to an uploaded object and the object will be overwritten after a namesake one is uploaded. 3. Objects can be downloaded by specifying the version ID. By default, the object of the latest version is downloaded if no version ID is specified. 4. Objects can be deleted by specifying version IDs. If an object is deleted with no version ID specified, the object is only attached with a delete marker whose version ID is **null**. Objects with version ID **null** are physically deleted. 5. Except for delete markers, storage space occupied by objects with all version IDs is billed.	Suspended

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
```



```
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

// Enable versioning.
$resp = $obsClient->setBucketVersioning([
     'Bucket' => 'bucketname',
     'Status' => 'Enabled'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);

// Suspend versioning.
$resp = $obsClient->setBucketVersioningConfiguration([
            'Bucket' => 'bucketname',
            'Status' => 'Suspended'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
```

NOTE

Use the **Status** parameter to specify the versioning status of a bucket.

10.3 Viewing Versioning Status of a Bucket

You can call **ObsClient->getBucketVersioning** to view the versioning status of a bucket.

This example obtains the versioning status of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketVersioning([
     'Bucket' => 'bucketname'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
printf ("Status:%s\n", $resp ['Status']);
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

10.4 Obtaining a Versioning Object

You can call **ObsClient->getObject** to obtain an object version by specifying the version ID (**VersionId**).

This example downloads object **objectname** from bucket **bucketname** by specifying **VersionId**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getObject([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'VersionId' => 'versionid'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
printf("Object Content:\n");
// Obtain the object content.
echo $resp ['Body'];
```

NOTE

- If **VersionId** is not specified, the object of the latest version will be downloaded.
- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

10.5 Copying a Versioning Object

You can call **ObsClient->copyObject** to copy an object version by specifying its **versionId** in the **CopySource** parameter.

This example specifies **versionId** to copy **sourceobjectname** of the specified version from **sourcebucketname** to **destbucketname** as **destobjectname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard
```



```
coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->copyObject([
     'Bucket' => 'destbucketname',
     'Key' => 'destobjectname',
     // Set the version ID of the object to be copied.
     'CopySource' => 'sourcebucket/sourceobjectname?versionId=versionid'
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

10.6 Restoring a Specific Cold Object Version

You can call **ObsClient->restoreObject** to restore a Cold object version by specifying **VersionId**.

This example specifies **versionId** to restore Cold object **destobjectname** in **destbucketname** as a Standard object.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->restoreObject([
     'Bucket' => 'destbucketname',
     'Key' => 'destobjectname',
     'VersionId' => 'versionid',
     'Days' => 1,
     // Restore a versioned object at an expedited speed.
     'Tier' => ObsClient::RestoreTierExpedited
]);

printf ("RequestId:%s\n", $resp ['RequestId']);
```


CAUTION

To prolong the validity period of the Cold data restored, you can repeatedly restore the data, but you will be billed for each restoration. After a second restore, the validity period of Standard object copies will be prolonged, and you need to pay for storing these copies during the prolonged period.

10.7 Listing Versioning Objects

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

The following table describes the parameters involved in this API.

Parameter	Description
Prefix	Name prefix that the objects to be listed must contain
KeyMarker	Object name to start with when listing versioning objects in a bucket. All versioning objects whose names follow this parameter are listed in the lexicographical order.
MaxKeys	Maximum number of versioning objects returned. The value ranges from 1 to 1000. If the value is not in this range, 1000 versioning objects are returned by default.
Delimiter	Character used to group object names. If the object name contains the **Delimiter** parameter, the character string from the first character to the first delimiter in the object name is grouped under a single result element, **CommonPrefix**. (If a prefix is specified in the request, the prefix must be removed from the object name.)
VersionIdMarker	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 **KeyMarker**.

NOTE

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

Listing Versioning Objects in Simple Mode

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

```
// Import the dependency library.
require 'vendor/autoload.php';
```



```
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->listVersions ( [
     'Bucket' => 'bucketname'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
// Obtain versioning objects.
printf ( "Versions:\n" );
foreach ( $resp ['Versions'] as $index => $version ) {
     printf ( "Versions[%d]\n", $index + 1 );
     printf ( "Key:%s\n", $version ['Key'] );
     printf ( "VersionId:%s\n", $version ['VersionId'] );
     printf ( "IsLatest:%s\n", $version ['IsLatest'] );
     printf ( "LastModified:%s\n", $version ['LastModified'] );
     printf ( "ETag:%s\n", $version ['ETag'] );
     printf ( "Size:%s\n", $version ['Size'] );
     printf ( "Owner[ID]:%s\n", $version ['Owner'] ['ID'] );
     printf ( "StorageClass:%s\n", $version ['StorageClass'] );
}
// Obtain delete markers.
printf ( "DeleteMarkers:\n" );
foreach ( $resp ['DeleteMarkers'] as $index => $deleteMarker ) {
     printf ( "DeleteMarkers[%d]\n", $index + 1 );
     printf ( "Key:%s\n", $deleteMarker ['Key'] );
     printf ( "VersionId:%s\n", $deleteMarker ['VersionId'] );
     printf ( "LastModified:%s\n", $deleteMarker ['LastModified'] );
     printf ( "Owner[ID]:%s\n", $deleteMarker ['Owner'] ['ID'] );
}
```

NOTE

- Information about a maximum of 1000 versioning objects can be listed each time. If a bucket contains more than 1000 objects and **IsTruncated** is **true** in the returned result, not all versioning objects are listed. In such cases, you can use **NextKeyMarker** and **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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );
```

```
// List 100 versioning objects.
$resp = $obsClient->listVersions ( [
      'Bucket' => 'bucketname',
      'MaxKeys' => 100
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
// Obtain versioning objects.
printf ( "Versions:\n" );
foreach ( $resp ['Versions'] as $index => $version ) {
      printf ( "Versions[%d]\n", $index + 1 );
      printf ( "Key:%s\n", $version ['Key'] );
      printf ( "VersionId:%s\n", $version ['VersionId'] );
      printf ( "IsLatest:%s\n", $version ['IsLatest'] );
      printf ( "LastModified:%s\n", $version ['LastModified'] );
      printf ( "ETag:%s\n", $version ['ETag'] );
      printf ( "Size:%s\n", $version ['Size'] );
      printf ( "Owner[ID]:%s\n", $version ['Owner'] ['ID'] );
      printf ( "StorageClass:%s\n", $version ['StorageClass'] );
}
// Obtain delete markers.
printf ( "DeleteMarkers:\n" );
foreach ( $resp ['DeleteMarkers'] as $index => $deleteMarker ) {
      printf ( "DeleteMarkers[%d]\n", $index + 1 );
      printf ( "Key:%s\n", $deleteMarker ['Key'] );
      printf ( "VersionId:%s\n", $deleteMarker ['VersionId'] );
      printf ( "LastModified:%s\n", $deleteMarker ['LastModified'] );
      printf ( "Owner[ID]:%s\n", $deleteMarker ['Owner'] ['ID'] );
}
```

Listing Versioning Objects by Specifying a Prefix

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
     //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
     //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
     'secret' => getenv('SECRET_ACCESS_KEY'),
     'endpoint' => 'https://your-endpoint'
] );

// Set the prefix to prefix and the number to 100.
$resp = $obsClient->listVersions ( [
      'Bucket' => 'bucketname',
      'MaxKeys' => 100,
      'Prefix' => 'prefix'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
// Obtain versioning objects.
printf ( "Versions:\n" );
foreach ( $resp ['Versions'] as $index => $version ) {
      printf ( "Versions[%d]\n", $index + 1 );
      printf ( "Key:%s\n", $version ['Key'] );
      printf ( "VersionId:%s\n", $version ['VersionId'] );
      printf ( "IsLatest:%s\n", $version ['IsLatest'] );
      printf ( "LastModified:%s\n", $version ['LastModified'] );
      printf ( "ETag:%s\n", $version ['ETag'] );
      printf ( "Size:%s\n", $version ['Size'] );
      printf ( "Owner[ID]:%s\n", $version ['Owner'] ['ID'] );
       printf ( "StorageClass:%s\n", $version ['StorageClass'] );
}
// Obtain delete markers.
```



```
printf ( "DeleteMarkers:\n" );
foreach ( $resp ['DeleteMarkers'] as $index => $deleteMarker ) {
    printf ( "DeleteMarkers[%d]\n", $index + 1 );
    printf ( "Key:%s\n", $deleteMarker ['Key'] );
    printf ( "VersionId:%s\n", $deleteMarker ['VersionId'] );
    printf ( "LastModified:%s\n", $deleteMarker ['LastModified'] );
    printf ( "Owner[ID]:%s\n", $deleteMarker ['Owner'] ['ID'] );
}
```

Listing Versioning Objects by Specifying the Start Position

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

// List 100 versioning objects whose names are following test in lexicographical order.
$resp = $obsClient->listVersions ( [
    'Bucket' => 'bucketname',
    'MaxKeys' => 100,
    'Marker' => 'test'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
// Obtain versioning objects.
printf ( "Versions:\n" );
foreach ( $resp ['Versions'] as $index => $version ) {
    printf ( "Versions[%d]\n", $index + 1 );
    printf ( "Key:%s\n", $version ['Key'] );
    printf ( "VersionId:%s\n", $version ['VersionId'] );
    printf ( "IsLatest:%s\n", $version ['IsLatest'] );
    printf ( "LastModified:%s\n", $version ['LastModified'] );
    printf ( "ETag:%s\n", $version ['ETag'] );
    printf ( "Size:%s\n", $version ['Size'] );
    printf ( "Owner[ID]:%s\n", $version ['Owner'] ['ID'] );
    printf ( "StorageClass:%s\n", $version ['StorageClass'] );
}
// Obtain delete markers.
printf ( "DeleteMarkers:\n" );
foreach ( $resp ['DeleteMarkers'] as $index => $deleteMarker ) {
    printf ( "DeleteMarkers[%d]\n", $index + 1 );
    printf ( "Key:%s\n", $deleteMarker ['Key'] );
    printf ( "VersionId:%s\n", $deleteMarker ['VersionId'] );
    printf ( "LastModified:%s\n", $deleteMarker ['LastModified'] );
    printf ( "Owner[ID]:%s\n", $deleteMarker ['Owner'] ['ID'] );
}
```

Listing All Versioning Objects in Paging Mode

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
```



```
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$keyMarker = null;
$versionIdMarker = null;

do {
     $resp = $obsClient->listVersions ( [
          'Bucket' => 'bucketname',
          'MaxKeys' => 100,
          'Marker' => $keyMarker,
          'VersionIdMarker' => $versionIdMarker
     ] );
     printf ( "RequestId:%s\n", $resp ['RequestId'] );
     // Obtain versioning objects.
     printf ( "Versions:\n" );
     foreach ( $resp ['Versions'] as $index => $version ) {
          printf ( "Versions[%d]\n", $index + 1 );
          printf ( "Key:%s\n", $version ['Key'] );
          printf ( "VersionId:%s\n", $version ['VersionId'] );
          printf ( "IsLatest:%s\n", $version ['IsLatest'] );
          printf ( "LastModified:%s\n", $version ['LastModified'] );
          printf ( "ETag:%s\n", $version ['ETag'] );
          printf ( "Size:%s\n", $version ['Size'] );
          printf ( "Owner[ID]:%s\n", $version ['Owner'] ['ID'] );
          printf ( "StorageClass:%s\n", $version ['StorageClass'] );
     }
     // Obtain delete markers.
     printf ( "DeleteMarkers:\n" );
     foreach ( $resp ['DeleteMarkers'] as $index => $deleteMarker ) {
          printf ( "DeleteMarkers[%d]\n", $index + 1 );
          printf ( "Key:%s\n", $deleteMarker ['Key'] );
          printf ( "VersionId:%s\n", $deleteMarker ['VersionId'] );
          printf ( "LastModified:%s\n", $deleteMarker ['LastModified'] );
          printf ( "Owner[ID]:%s\n", $deleteMarker ['Owner'] ['ID'] );
     }

     $keyMarker = $resp['NextKeyMarker'];
     $versionIdMarker = $resp['NextVersionIdMarker'];
}while($resp['IsTruncated']);
```

Listing All Versioning Objects in a Folder

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

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
```



```
    'endpoint' => 'https://your-endpoint'
] );

$keyMarker = null;
$versionIdMarker = null;

do {
    $resp = $obsClient->listVersions ( [
        'Bucket' => 'bucketname',
        'MaxKeys' => 100,
        'Marker' => $keyMarker,
        'VersionIdMarker' => $versionIdMarker,
        // Set the prefix of the folders to dir/.
        'Prefix' => 'dir/'
    ] );
    printf ( "RequestId:%s\n", $resp ['RequestId'] );
    // Obtain versioning objects.
    printf ( "Versions:\n" );
    foreach ( $resp ['Versions'] as $index => $version ) {
        printf ( "Versions[%d]\n", $index + 1 );
        printf ( "Key:%s\n", $version ['Key'] );
        printf ( "VersionId:%s\n", $version ['VersionId'] );
        printf ( "IsLatest:%s\n", $version ['IsLatest'] );
        printf ( "LastModified:%s\n", $version ['LastModified'] );
        printf ( "ETag:%s\n", $version ['ETag'] );
        printf ( "Size:%s\n", $version ['Size'] );
        printf ( "Owner[ID]:%s\n", $version ['Owner'] ['ID'] );
        printf ( "StorageClass:%s\n", $version ['StorageClass'] );
    }
    // Obtain delete markers.
    printf ( "DeleteMarkers:\n" );
    foreach ( $resp ['DeleteMarkers'] as $index => $deleteMarker ) {
        printf ( "DeleteMarkers[%d]\n", $index + 1 );
        printf ( "Key:%s\n", $deleteMarker ['Key'] );
        printf ( "VersionId:%s\n", $deleteMarker ['VersionId'] );
        printf ( "LastModified:%s\n", $deleteMarker ['LastModified'] );
        printf ( "Owner[ID]:%s\n", $deleteMarker ['Owner'] ['ID'] );
    }

    $keyMarker = $resp['NextKeyMarker'];
    $versionIdMarker = $resp['NextVersionIdMarker'];
}while($resp['IsTruncated']);
```

Listing All Versioning Objects According to Folders in a Bucket

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

function listVersionsByPrefix($commonPrefixes){
    global $obsClient;
    foreach ($commonPrefixes as $commonPrefix){
        $resp = $obsClient->listVersions ( [
                    'Bucket' => 'bucketname',
                    // Set folder isolators to slashes (/).
                    'Delimiter' => '/',
```

```
                    'Prefix' => $commonPrefix['Prefix']
            ] );
            printf ( "Objects in folder [%s]:\n ", $commonPrefix['Prefix']);
            // Obtain versioning objects.
            printf ( "Versions:\n" );
            foreach ( $resp ['Versions'] as $index => $version ) {
                printf ( "Versions[%d]\n", $index + 1 );
                printf ( "Key:%s\n", $version ['Key'] );
                printf ( "VersionId:%s\n", $version ['VersionId'] );
            }
            // Obtain delete markers.
            printf ( "DeleteMarkers:\n" );
            foreach ( $resp ['DeleteMarkers'] as $index => $deleteMarker ) {
                printf ( "DeleteMarkers[%d]\n", $index + 1 );
                printf ( "Key:%s\n", $deleteMarker ['Key'] );
                printf ( "VersionId:%s\n", $deleteMarker ['VersionId'] );
            }
            printf("\n");
            listVersionsByPrefix($resp['CommonPrefixes']);
        }
}

$resp = $obsClient->listVersions ( [
        'Bucket' => 'bucketname',
        // Set folder isolators to slashes (/).
        'Delimiter' => '/'
] );
printf ( "Objects in the root directory:\n ");
// Obtain versioning objects.
printf ( "Versions:\n" );
foreach ( $resp ['Versions'] as $index => $version ) {
        printf ( "Versions[%d]\n", $index + 1 );
        printf ( "Key:%s\n", $version ['Key'] );
        printf ( "VersionId:%s\n", $version ['VersionId'] );
}
            // Obtain delete markers.
printf ( "DeleteMarkers:\n" );
foreach ( $resp ['DeleteMarkers'] as $index => $deleteMarker ) {
        printf ( "DeleteMarkers[%d]\n", $index + 1 );
        printf ( "Key:%s\n", $deleteMarker ['Key'] );
        printf ( "VersionId:%s\n", $deleteMarker ['VersionId'] );
}
printf("\n");

listVersionsByPrefix($resp['CommonPrefixes']);
```

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, **Versions** includes the versioning objects in the folder, **DeleteMarkers** includes the delete markers in the folder, and **CommonPrefixes** includes the sub-folders in the folder.

10.8 Setting or Obtaining a Versioning Object ACL

Setting an ACL for an Object Version

You can call ObsClient->setObjectAcl to set the ACL for an object version by specifying the version ID (**VersionId**). Sample code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
```

```
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient -> setObjectAcl([
     'Bucket' => 'bucketname',
     'Key' => 'objectname',
     'VersionId' => 'versionid',
     // Set the object version ACL to public read by specifying a pre-defined ACL.
     'ACL' => ObsClient::AclPublicRead
]);

printf("RequestId:%s\n", $resp['RequestId']);

$resp = $obsClient -> setObjectAcl([
            'Bucket' => 'bucketname',
            'Key' => 'objectname',
            'VersionId' => 'versionid',
            // Set the object owner.
            'Owner' => ['ID' => 'ownerid'],
            'Grants' => [
                  // Grant the READ permission to all users.
                  ['Grantee' => ['Type' => 'Group', 'URI' => ObsClient::GroupAllUsers], 'Permission' => ObsClient::PermissionRead],
            ]
]);

printf("RequestId:%s\n", $resp['RequestId']);
```

NOTE

- Use the **Owner** parameter to specify the object owner and the **Grants** parameter to specify information about the authorized users.
- The owner or grantee ID required in the ACL indicates an account ID, which can be viewed on the **My Credentials** page of OBS Console.
- OBS buckets support the following grantee group:
  - All users: ObsClient::GroupAllUsers

Obtaining the ACL of an Object Version

You can call ObsClient->getObjectAcl to obtain the ACL of an object version by specifying the version ID (**VersionId**). Sample code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
```

```
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getObjectAcl ( [
      'Bucket' => 'bucketname',
      'Key' => 'objectname',
      'VersionId' => 'versionid'
] );

printf ( "RequestId:%s\n", $resp ['RequestId'] );
printf ( "Owner[ID]:%s\n", $resp ['Owner'] ['ID'] );
foreach ( $resp ['Grants'] as $index => $grant ) {
      printf ( "Grants[%d]\n", $index + 1 );
      printf ( "Grantee[ID]:%s\n", $grant ['Grantee'] ['ID'] );
      printf ( "Grantee[URI]:%s\n", $grant ['Grantee'] ['URI'] );
      printf ( "Permission:%s\n", $grant ['Permission'] );
}
```

10.9 Deleting Versioning Objects

Deleting a Single Versioning Object

You can call **ObsClient->deleteObject** to delete an object version by specifying the version ID (**VersionId**).

This example deletes object **objectname** from bucket **bucketname** by specifying **VersionId**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->deleteObject ( [
      'Bucket' => 'bucketname',
      'Key' => 'objectname',
      'VersionId' => 'versionid'
] );

printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

Batch Deleting Versioning Objects

You can call **ObsClient->deleteObjects** to batch delete specific versions of an object by passing the **VersionId** value of each version to delete.

This example deletes objects **objectname1** and **objectname2** from bucket **bucketname** in a batch by specifying their version IDs.

The example code is as follows:


```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->deleteObjects ( [
            'Bucket' => 'bucketname',
            // Set the response mode to verbose.
            'Quiet' => false,
            'Objects' => [
                        [
                                    'Key' => 'objectname1',
                                    'VersionId' => 'versionid1'
                        ],
                        [
                                    'Key' => 'objectname2',
                                    'VersionId' => 'versionid2'
                        ]
            ]
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
// Obtain the successfully deleted objects.
printf ( "Deleteds:\n" );
foreach ( $resp ['Deleteds'] as $index => $deleted ) {
      printf ( "Deleteds[%d]", $index + 1 );
      printf ( "Key:%s\n", $deleted ['Key'] );
      printf ( "VersionId:%s\n", $deleted ['VersionId'] );
      printf ( "DeleteMarker:%s\n", $deleted ['DeleteMarker'] );
      printf ( "DeleteMarkerVersionId:%s\n", $deleted ['DeleteMarkerVersionId'] );
}
// Obtain the objects that failed to be deleted.
printf ( "Errors:\n" );
foreach ( $resp ['Errors'] as $index => $error ) {
      printf ( "Errors[%d]", $index + 1 );
      printf ( "Key:%s\n", $error ['Key'] );
      printf ( "VersionId:%s\n", $$error ['VersionId'] );
      printf ( "Code:%s\n", $error ['Code'] );
      printf ( "Message:%s\n", $error ['Message'] );
}
```


11 Lifecycle Management

11.1 Lifecycle Management Overview

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

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

NOTE

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

11.2 Setting Lifecycle Rules

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

Setting an Object Transition Policy

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';

// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'

] );

$resp = $obsClient->setBucketLifecycle ( [
     'Bucket' => 'bucketname',
     'Rules' => [
                [
                    'ID' => 'rule1',
                    'Prefix' => 'prefix1',
                    'Status' => 'Enabled',
                    'Transitions' => [
                        [
                            'StorageClass' => ObsClient::StorageClassWarm,
                            'Days' => 30
                        ]
                    ],
                    'NoncurrentVersionTransitions' => [
                        [
                            'StorageClass' => ObsClient::StorageClassCold,
                            'NoncurrentDays' => 30
                        ]
                    ]
                ],
                [
                    'ID' => 'rule2',
                    'Prefix' => 'prefix2',
                    'Status' => 'Enabled',
                    'Transitions' => [
                            [
                                'StorageClass' => ObsClient::StorageClassWarm,
                                'Date' => '2018-12-31T00:00:00Z'
                            ]
                    ]
                ]
    ]
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

Setting an Object Expiration Time

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
```

```
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint',
] );

$resp = $obsClient->setBucketLifecycle ( [
     'Bucket' => 'bucketname',
     'Rules' => [
                [
                    'ID' => 'rule1',
                    'Prefix' => 'prefix1',
                    'Status' => 'Enabled',
                    // Specify that objects whose names contain the prefix will expire after being created for 60 days.
                    'Expiration' => [
                                'Days' => 60
                    ],
                    // Specify that objects whose names contain the prefix will expire after becoming noncurrent versions for 60 days.
                    'NoncurrentVersionExpiration' => [
                                'NoncurrentDays' => 60
                    ]
                ],
                [
                    'ID' => 'rule2',
                    'Prefix' => 'prefix2',
                    'Status' => 'Enabled',
                    // Specify a date when the objects whose name contain the prefix will expire. The value must conform to the ISO8601 standards and must be at 00:00 (UTC time).
                    'Expiration' => [
                                'Date' => '2018-12-31T00:00:00Z'
                    ]
                ]
    ]
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

Use the **Rules** parameter to specify the lifecycle rules for a bucket.

11.3 Viewing Lifecycle Rules

You can call **ObsClient->getBucketLifecycle** to view lifecycle rules of a bucket.

This example views the lifecycle configuration of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
```

```
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'

] );

$resp = $obsClient->getBucketLifecycle ( [
      'Bucket' => 'bucketname'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
foreach ($resp['Rules'] as $index => $rule) {
    printf("Rules[%d]\n", $index + 1);
    printf("ID:%s\n", $rule['ID']);
    printf("Prefix:%s\n", $rule['Prefix']);
    printf("Status:%s\n", $rule['Status']);
    foreach ($rule['Transitions'] as $i => $transition) {
        printf("[Transitions][$i][Date]:%s,[Transitions][$i][StorageClass]:%s\n", $transition['Date'],
$transition['StorageClass']);
    }
    printf("Expiration[Days]:%s\n", $rule['Expiration']['Days']);
    printf("Expiration[Date]:%s\n", $rule['Expiration']['Date']);
    foreach ($rule['NoncurrentVersionTransitions'] as $i => $noncurrentVersionTransition) {
        printf("[NoncurrentVersionTransitions][$i][NoncurrentDays]:%d,[NoncurrentVersionTransitions][$i]
[StorageClass]:%s\n", $noncurrentVersionTransition['NoncurrentDays'],
$noncurrentVersionTransition['StorageClass']);
    }
    printf("NoncurrentVersionExpiration[NoncurrentDays]:%s\n", $rule['NoncurrentVersionExpiration']
['NoncurrentDays']);
}
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

11.4 Deleting Lifecycle Rules

You can call **ObsClient->deleteBucketLifecycle** to delete lifecycle rules of a bucket.

This example deletes the lifecycle configuration of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard
coding may result in leakage.
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->deleteBucketLifecycle ( [
      'Bucket' => 'bucketname'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

12 CORS

12.1 CORS Overview

Cross-origin access refers to access between different domains. Restricting cross-origin access is a browser policy for security purposes, that is, the same-origin policy.

Due to this same-origin policy, JavaScript in origin A cannot operate objects in origin B or C.

The same-origin policy requires the protocols, domain names (or IP addresses), and ports are all the same. If the protocols, domain names, and ports (if specified) of the two web pages are the same, the two web pages are in the same origin.

CORS allows web application programs in one origin to access resources in another. OBS provides developers with APIs for facilitating cross-origin resource access.

12.2 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );
```



```
$resp = $obsClient->setBucketCors ( [
      'Bucket' => 'bucketname',
      'CorsRules' => [
                  [
                        'ID' => 'rule1',
                        //Specify the request method, which can be GET, PUT, DELETE, POST, or HEAD.
                        'AllowedMethod' => [ 'GET','HEAD','PUT'],
                        //Specify the origin of the cross-domain 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 response headers that users can access using application programs.
                        'ExposeHeader' => ['x-obs-expose-header'],
                        // Specify the browser's cache time of the returned results of OPTIONS requests for
specific resources, in seconds.
                        'MaxAgeSeconds' => 60
                  ]
      ]
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

- Use the **CorsRules** parameter to set CORS rules for a bucket.
- Both **AllowedOrigin** and **AllowedHeader** can contain one wildcard character (*). The wildcard character (*) indicates that all origins or headers are allowed.

12.3 Viewing CORS Rules

You can call **ObsClient->getBucketCors** to view CORS rules of a bucket.

This example views the CORS rule of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
      //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard
coding may result in leakage.
      //Obtain an AK/SK pair on the management console.       'key' => getenv('ACCESS_KEY_ID'),
      'secret' => getenv('SECRET_ACCESS_KEY'),
      'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketCors ( [
       'Bucket' => 'bucketname'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
foreach ( $resp ['CorsRules'] as $index => $rule ) {
       printf ( "CorsRule[%d]\n", $index + 1 );
       printf ( "ID:%s\n", $rule ['ID'] );
       printf ( "MaxAgeSeconds:%s\n", $rule ['MaxAgeSeconds'] );
       printf ( "AllowedMethod:%s\n", print_r ( $rule ['AllowedMethod'], true ) );
       printf ( "AllowedOrigin:%s\n", print_r ( $rule ['AllowedOrigin'], true ) );
       printf ( "AllowedHeader:%s\n", print_r ( $rule ['AllowedHeader'], true ) );
       printf ( "ExposeHeader:%s\n", print_r ( $rule ['ExposeHeader'], true ) );
}
```


NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

12.4 Deleting CORS Rules

You can call **ObsClient->deleteBucketCors** to delete CORS rules of a bucket.

This example deletes the CORS rule of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.     'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->deleteBucketCors ( [
     'Bucket' => 'bucketname'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

13 Access Logging

13.1 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.

You can enable OBS logging for bucket analysis or audit purposes. With access logs, a bucket owner can analyze the characteristics, types, or trends of requests sent to the bucket.

With logging enabled, OBS automatically logs access requests for the bucket and writes the generated log files into a specified bucket.

You need to specify a bucket for storing log files when enabling logging for a bucket. Log files can be stored in any bucket you own in the region where the logged bucket is, including the logged bucket itself.

13.2 Enabling Bucket Logging

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

NOTE

- The source bucket and target bucket of logging must be in the same region.
- If the bucket is in the OBS Warm or Cold storage class, it cannot be used as the target bucket.

Enabling Bucket Logging

Sample code:

```
// Import the dependency library.
require'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
useObs\ObsClient;
```



```
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );
$targetBucketName = 'targetbucketname';
// Configure logging for the bucket.
$resp = $obsClient -> setBucketLogging( [
    'Bucket' => 'bucketname',
    // Name of the OBS agency created by the owner of the target bucket on IAM.
    'Agency' => 'Agency name',
    'LoggingEnabled'=> [
        'TargetBucket' => $targetBucketName,
        'TargetPrefix '=> 'prefix',
    ]
] );
printf("RequestId:%s\n", $resp['RequestId']);
```

NOTE

Use the **LoggingEnabled** parameter to configure logging for a bucket.

Setting ACLs for Objects to Be Logged

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during the installation with source code.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an ObsClient instance.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );
;

$targetBucketName = 'targetbucketname';

// Configure logging for the bucket.
$resp = $obsClient->setBucketLogging ( [
     'Bucket' => 'bucketname',
     // Name of the OBS agency created by the owner of the target bucket on IAM.
     'Agency' => 'Agency name',
     'LoggingEnabled' => [
          'TargetBucket' => $targetBucketName,
          'TargetPrefix' => 'prefix',
          'TargetGrants' => [
                    // Grant all users the READ permission on the logs.
                    [
                         'Grantee' => [
                                   'URI' => ObsClient::GroupAllUsers,
                                   'Type' => 'Group'
                         ],
                         'Permission' => ObsClient::PermissionRead
                    ],
          ]
    ]
```

```
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

13.3 Viewing Bucket Logging Settings

You can call **ObsClient->getBucketLogging** to view the logging settings of a bucket.

This example views the logging configuration of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.
    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketLogging ( [
     'Bucket' => 'bucketname'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
printf ( "LoggingEnabled[TargetBucket]:%s\n", $resp ['LoggingEnabled'] ['TargetBucket'] );
printf ( "LoggingEnabled[TargetPrefix]:%s\n", $resp ['LoggingEnabled'] ['TargetPrefix'] );
if (is_array ( $resp ['LoggingEnabled'] ['TargetGrants'] )) {
    foreach ( $resp ['LoggingEnabled'] ['TargetGrants'] as $index => $grant ) {
        printf ( "TargetGrants[%d]", $index + 1 );
        printf ( "Grantee[ID]:%s\n", $grant ['Grantee'] ['ID'] );
        printf ( "Grantee[URI]:%s\n", $grant ['Grantee'] ['URI'] );
        printf ( "Permission:%s\n", $grant ['Permission'] );
    }
}
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

13.4 Disabling Bucket Logging

To disable logging for a bucket is to call **ObsClient->setBucketLogging** to delete the logging configuration.

This example disables logging for bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
```



```
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.      'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->setBucketLogging ( [
     'Bucket' => 'bucketname',
     'LoggingEnabled' => []
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

14 Static Website Hosting

14.1 Static Website Hosting Overview

Static websites typically only contain static web pages and some scripts that can run on clients (such as JavaScript and Flash). In contrast, dynamic websites depend on scripts that need to be processed on the server side, including PHP, JSP, and ASP.Net.

To host your static website on OBS, you can upload static website files to your bucket as objects, configure the public read permission for the objects, and then configure static website hosting for your bucket.

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.

14.2 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 object ACL to public read.

Step 3 Access the object using a browser.

----End

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
```



```
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

// Upload an object.
$resp = $obsClient->putObject ( [
     'Bucket' => 'bucketname',
     'Key' => 'test.html',
     'Body' => '<html><header></header><body><h1>Hello OBS</h1></body></html>',
     // Set the MIME type for the object.
     'ContentType' => 'text/html',
     // Set the object ACL to public read.
     'ACL' => ObsClient::AclPublicRead
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

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

14.3 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 dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->setBucketWebsite( [
     'Bucket' => 'bucketname',
     // Configure the default homepage.
     'IndexDocument' => ['Suffix' => 'index.html'],
     // Configure the error pages.
     'ErrorDocument' => ['Key' => 'error.html']
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

Configuring Redirection Rules

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
```

```
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->setBucketWebsite( [
     'Bucket' => 'bucketname',
     // Configure the default homepage.
     'IndexDocument' => ['Suffix' => 'index.html'],
     // Configure the error pages.
     'ErrorDocument' => ['Key' => 'error.html'],
     // Set redirection rules.
     'RoutingRules' => [
                ['Condition' => ['HttpErrorCodeReturnedEquals' => 404], 'Redirect' => ['Protocol' => 'http', 'ReplaceKeyWith' => 'NotFound.html']],
                ['Condition' => ['HttpErrorCodeReturnedEquals' => 404], 'Redirect' => ['Protocol' => 'https', 'ReplaceKeyWith' => 'test.html']]
     ]
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

Use the **RoutingRules** parameter to specify redirection rules for a bucket.

Configuring Redirection for All Requests

Sample code:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.    'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->setBucketWebsite( [
     'Bucket' => 'bucketname',
     'RedirectAllRequestsTo' => ['HostName' => 'www.example.com', 'Protocol' => 'http']
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

Use the **RedirectAllRequestsTo** parameter to set redirection rules for all requests for accessing a bucket.

14.4 Viewing Website Hosting Settings

You can call **ObsClient->getBucketWebsite** to view the hosting settings of a bucket.

This example views the hosting configuration of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.     'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );

$resp = $obsClient->getBucketWebsite ( [
     'Bucket' => 'bucketname'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
printf ( "IndexDocument[Suffix]:%s\n", $resp ['IndexDocument'] ['Suffix'] );
printf ( "ErrorDocument[Key]:%s\n", $resp ['ErrorDocument'] ['Key'] );
foreach ( $resp ['RoutingRules'] as $index => $routingRule ) {
     printf ( "RoutingRules[%d]\n", $index + 1 );
     printf ( "Condition[HttpErrorCodeReturnedEquals]:%s\n", $routingRule ['Condition'] ['HttpErrorCodeReturnedEquals'] );
     printf ( "Condition[KeyPrefixEquals]:%s\n", $routingRule ['Condition'] ['KeyPrefixEquals'] );
     printf ( "Redirect[HostName]:%s\n", $routingRule ['Redirect'] ['HostName'] );
     printf ( "Redirect[Protocol]:%s\n", $routingRule ['Redirect'] ['Protocol'] );
     printf ( "Redirect[HttpRedirectCode]:%s\n", $routingRule ['Redirect'] ['HttpRedirectCode'] );
     printf ( "Redirect[ReplaceKeyPrefixWith]:%s\n", $routingRule ['Redirect'] ['ReplaceKeyPrefixWith'] );
     printf ( "Redirect[ReplaceKeyWith]:%s\n", $routingRule ['Redirect'] ['ReplaceKeyWith'] );
}
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

14.5 Deleting Website Hosting Settings

You can call **ObsClient->deleteBucketWebsite** to delete the hosting settings of a bucket.

This example deletes the hosting configuration of bucket **bucketname**.

The example code is as follows:

```
// Import the dependency library.
require 'vendor/autoload.php';
// Import the SDK code library during source code installation.
// require 'obs-autoloader.php';
// Declare the namespace.
use Obs\ObsClient;
// Create an instance of ObsClient.
$obsClient = new ObsClient ( [
    //Obtain an AK/SK pair using environment variables or import the AK/SK pair in other ways. Using hard coding may result in leakage.
    //Obtain an AK/SK pair on the management console.     'key' => getenv('ACCESS_KEY_ID'),
    'secret' => getenv('SECRET_ACCESS_KEY'),
    'endpoint' => 'https://your-endpoint'
] );
```

```
$resp = $obsClient->deleteBucketWebsite ( [
    'Bucket' => 'bucketname'
] );
printf ( "RequestId:%s\n", $resp ['RequestId'] );
```

NOTE

- To handle the error codes possibly returned during the operation, see **OBS Server-Side Error Codes**.

15 Troubleshooting

15.1 OBS Server-Side Error Codes

If the OBS server encounters an error when processing a request, a response containing the error code and error description is returned. The following table lists details about each error code and HTTP status code.

Error Code	Description	HTTP Status Code
AccessDenied	Access denied.	403 Forbidden
AccessForbidden	Insufficient permission.	403 Forbidden
AccountProblem	Your account is abnormal (for example, it has been expired or frozen).	403 Forbidden
AllAccessDisabled	You have no permission to perform the operation.	403 Forbidden
AmbiguousGrantByEmailAddress	The provided email address is associated with more than one account.	400 Bad Request
BadDigest	The specified value of Content-MD5 does not match the value received by OBS.	400 Bad Request
BadDomainName	Invalid domain name.	400 Bad Request
BadRequest	Invalid request parameters.	400 Bad Request

Error Code	Description	HTTP Status Code
BucketAlreadyExists	The requested bucket name already exists. The bucket namespace is shared by all users of OBS. Specify a different name and retry.	409 Conflict
BucketAlreadyOwnedByYou	Your previous request for creating the named bucket succeeded and you already own it.	409 Conflict
BucketNotEmpty	The bucket that you tried to delete is not empty.	409 Conflict
CredentialsNotSupported	This request does not support security credentials.	400 Bad Request
CustomDomainAreadyExist	The configured domain already exists.	400 Bad Request
CustomDomainNotExist	The domain to be operated does not exist.	400 Bad Request
DeregisterUserId	The user has been deregistered.	403 Forbidden
EntityTooSmall	The size of the object to be uploaded is smaller than the lower limit.	400 Bad Request
EntityTooLarge	The size of the object to be uploaded exceeds the upper limit.	400 Bad Request
FrozenUserId	The user has been frozen.	403 Forbidden
IllegalVersioningConfigurationException	The **Versioning** configuration specified in the request is invalid.	400 Bad Request
IllegalLocationConstraintException	The configured region limitation is inconsistent with the region where it resides.	400 Bad Request
InArrearOrInsufficientBalance	The user has no permission to perform some operations due to being in arrears or insufficient funds.	403 Forbidden

Error Code	Description	HTTP Status Code
IncompleteBody	Incomplete request body.	400 Bad Request
IncorrectNumberOfFilesInPostRequest	Each POST request must contain one file to be uploaded.	400 Bad Request
InlineDataTooLarge	The size of inline data exceeds the upper limit.	400 Bad Request
InsufficientStorageSpace	Insufficient storage space.	403 Forbidden
InternalError	An internal error occurs. Retry later.	500 Internal Server Error
InvalidAccessKeyId	The access key ID provided by the customer does not exist in the system.	403 Forbidden
InvalidAddressingHeader	The anonymous role must be specified.	N/A
InvalidArgument	Invalid parameter.	400 Bad Request
InvalidBucketName	The specified bucket name in the request is invalid.	400 Bad Request
InvalidBucket	The bucket to be accessed does not exist.	400 Bad Request
InvalidBucketState	Invalid bucket status.	409 Conflict
InvalidBucketStoragePolicy	An invalid new policy is specified during bucket policy modification.	400 Bad Request
InvalidDigest	The specified **Content-MD5** in the HTTP header is invalid.	400 Bad Request
InvalidEncryptionAlgorithmError	Incorrect encryption algorithm.	400 Bad Request
InvalidLocationConstraint	The location specified during bucket creation is invalid.	400 Bad Request
InvalidPart	One or more specified parts are not found. The parts may not be uploaded or the specified entity tags (ETags) do not match the parts' ETags.	400 Bad Request

Error Code	Description	HTTP Status Code
InvalidPartOrder	Parts are not listed in ascending order by part number.	400 Bad Request
InvalidPayer	All accesses to this object are disabled.	403 Forbidden
InvalidPolicyDocument	The content of the form does not meet the conditions specified in the policy document.	400 Bad Request
InvalidRange	The requested range is invalid.	416 Client Requested Range Not Satisfiable
InvalidRedirectLocation	Invalid redirect location.	400 Bad Request
InvalidRequest	Invalid request.	400 Bad Request
InvalidRequestBody	Invalid POST request body.	400 Bad Request
InvalidSecurity	Invalid security credentials.	403 Forbidden
InvalidStorageClass	The specified storage class is invalid.	400 Bad Request
InvalidTargetBucketForLogging	The delivery group has no ACL permission for the target bucket.	400 Bad Request
InvalidURI	Cannot resolve the specified uniform resource identifier (URI).	400 Bad Request
KeyTooLong	The provided key is too long.	400 Bad Request
MalformedACLError	The provided XML has bad syntax or does not meet the format requirements.	400 Bad Request
MalformedError	The XML format in the request is incorrect.	400 Bad Request
MalformedLoggingStatus	The XML format of **Logging** is incorrect.	400 Bad Request
MalformedPolicy	The bucket policy failed the check.	400 Bad Request

Error Code	Description	HTTP Status Code
MalformedPOSTRequest	The body of the POST request is in an incorrect format.	400 Bad Request
MalformedQuotaError	The Quota XML format is incorrect.	400 Bad Request
MalformedXML	This error code is returned after you send an XML file in incorrect format, stating "The XML you provided was not well-formed or did not validate against our published schema."	400 Bad Request
MaxMessageLengthExceeded	The request is too long.	400 Bad Request
MaxPostPreDataLengthExceededError	The POST request fields prior to the file to be uploaded are too large.	400 Bad Request
MetadataTooLarge	The size of the metadata header has exceeded the upper limit.	400 Bad Request
MethodNotAllowed	The specified method is not allowed against the requested resource.	405 Method Not Allowed
MissingContentLength	The HTTP header **Content-Length** is not provided.	411 Length Required
MissingRegion	No region in the request and no default region in the system.	400 Bad Request
MissingRequestBodyError	This error code is returned after you send an empty XML file, stating "Request body is empty."	400 Bad Request
MissingRequiredHeader	Required headers missing in the request.	400 Bad Request
MissingSecurityHeader	A required header is not provided.	400 Bad Request
NoSuchBucket	The specified bucket does not exist.	404 Not Found
NoSuchBucketPolicy	No bucket policy exists.	404 Not Found

Error Code	Description	HTTP Status Code
NoSuchCORSConfiguration	No CORS configuration exists.	404 Not Found
NoSuchCustomDomain	The requested user domain does not exist.	404 Not Found
NoSuchKey	The specified key does not exist.	404 Not Found
NoSuchLifecycleConfiguration	The requested **Lifecycle** does not exist.	404 Not Found
NoSuchPolicy	The specified policy name does not exist.	404 Not Found
NoSuchUpload	The specified multipart upload does not exist. The upload ID does not exist or the multipart upload has been aborted or completed.	404 Not Found
NoSuchVersion	The specified version ID does not match any existing version.	404 Not Found
NoSuchWebsiteConfiguration	The requested website does not exist.	404 Not Found
NotImplemented	The provided header implies a function that is unavailable.	501 Not Implemented
NotSignedUp	Your account has not registered in the system.	403 Forbidden
OperationAborted	A conflicting operation is being performed on this resource. Retry later.	409 Conflict
PermanentRedirect	The requested bucket has been permanently redirected to a new URL. All future requests must be sent to the new URL.	301 Moved Permanently
PreconditionFailed	At least one of the specified preconditions is not met.	412 Precondition Failed
Redirect	The request is temporarily redirected.	307 Moved Temporarily

Error Code	Description	HTTP Status Code
RequestIsNotMultiPartContent	A bucket POST request must contain an enclosure-type multipart or the form-data.	400 Bad Request
RequestTimeout	The socket connection to the server has no reads or writes within the timeout period.	400 Bad Request
RequestTimeTooSkewed	The request time and the server's time differ a lot.	403 Forbidden
RequestTorrentOfBucketError	Requesting the bucket's torrent file is not allowed.	400 Bad Request
ServiceNotImplemented	The request method is not implemented by the server.	501 Not Implemented
ServiceNotSupported	The request method is not supported by the server.	409 Conflict
ServiceUnavailable	The server is overloaded or has internal errors.	503 Service Unavailable
SignatureDoesNotMatch	The provided signature does not match the signature calculated by OBS. Check your AK and SK and signature calculation method.	403 Forbidden
SlowDown	Too frequent requests. Reduce your request frequency.	503 Service Unavailable
System Capacity Not enough	Insufficient system space.	403 Forbidden
TooManyCustomDomains	Too many user domains are configured.	400 Bad Request
TemporaryRedirect	The request is redirected to the bucket while the domain name server (DNS) is being updated.	307 Moved Temporarily
TooManyBuckets	You have attempted to create more buckets than allowed.	400 Bad Request
TooManyObjectCopied	The number of copied users' objects exceeds the upper limit.	400 Bad Request

Error Code	Description	HTTP Status Code
TooManyWrongSignature	The request is rejected due to high-frequency errors.	400 Bad Request
UnexpectedContent	This request does not support content.	400 Bad Request
UnresolvableGrantByEmailAddress	The provided email address does not match any recorded accounts.	400 Bad Request
UserKeyMustBeSpecified	The user's AK is not carried in the request.	400 Bad Request
WebsiteRedirect	The website request lacks **bucketName**.	301 Moved Permanently
RestoreAlreadyInProgress	The archive objects are being restored. The request conflicts with another one.	409 Conflict
ObjectHasAlreadyRestored	The objects have been restored and the retention period of the objects cannot be shortened.	409 Conflict
InvalidObjectState		403 Forbidden

15.2 SDK Custom Exceptions

SDK custom exceptions (**Obs\ObsException**), thrown by **ObsClient**, are inherited from class **\RuntimeException**. Exceptions are usually OBS server errors, including **OBS error codes** and error information. This facilitates users to locate problems and troubleshot faults.

Obs\ObsException contains the following error information:

- **ObsException->getStatusCode**: HTTP status code
- **ObsException->getExceptionCode**: Error code returned by the OBS server
- **ObsException->getExceptionMessage**: Error description returned by the OBS server
- **ObsException->getRequestId**: Request ID returned by the OBS server
- **ObsException->getHostId**: Requested server ID.
- **ObsException->getResponse**: HTTP response object
- **ObsException->getRequest**: HTTP request object

15.3 SDK Common Result Objects

After an API is called using an instance of **ObsClient**, view whether an exception is thrown. If no, an SDK common result object will be returned, indicating that the operation is successful. The following table lists the object content:

Field	Storage Class	Description
HttpStatusCode	integer	HTTP status code
Reason	string	Reason description
RequestId	string	Request ID returned by the OBS server
Other fields	See the *OBS PHP SDK API Reference*.	

15.4 Log Analysis

Log Configuration

OBS PHP SDK provides the logging function based on the monolog log library. You can call **ObsClient->initLog** to enable and configure logging. Sample code is as follows:

```
$obsClient -> initLog ([
    'FilePath' => './logs', // Set the log folder.
    'FileName' => 'eSDK-OBS-PHP.log', // Set the name for the log file.
    'MaxFiles' => 10, // Set the maximum number of log files that can be retained.
    'Level' => WARN  // Set the log level.
]);
```

NOTE

- The logging function is disabled by default. You need to enable it if needed.
- Use the **FilePath** parameter to specify the log file path. The path can be an absolute path or a relative path.

Log Format

The SDK log format is "Log time|Log level|File and code line number of the printed log|Log content". The following are example logs:

```
[2017-11-17 11:46:24][INFO][SendRequestTrait.php:376]: enter method createBucketAsync...
[2017-11-17 11:46:24][INFO][SendRequestTrait.php:525]: http request cost 97 ms
[2017-11-17 11:46:24][INFO][SendRequestTrait.php:538]: obsclient cost 155 ms to execute
createBucketAsync
```

Log Level

When current logs cannot be used to troubleshoot system faults, you can change the log level to obtain more information. SDK defines four types of integer constant corresponding to different log levels. You can obtain the most information in **DEBUG** logs and the least information in **ERROR** logs.

Log level description:

- **DEBUG** (100): Debugging level. If this level is set, all logs will be printed.
- **INFO** (200): Information level. If this level is set, logs at the **WARN** level and the time consumed for each HTTP/HTTPS request will be printed.
- **WARN** (300): Warning level. If this level is set, logs at the **ERROR** level and some critical events will be printed.
- **ERROR** (400): Error level. If this level is set, only error information will be printed.

15.5 Time Zone Configuration Failure

If an error (such as **Uncaught exception 'Exception' with message 'DateTime::__construct():**) is displayed when you use OBS PHP SDK for secondary development, the time zone configuration is incorrect. You can configure the time zone using either of the following methods:

1. Modify the **php.ini** file by adding **date.timezone = xxx** in the **[date]** tag, for example, **date.timezone = UTC**.
2. Call **date_default_timezone_set('xxx')** in the program to directly set the time zone.

16 FAQs

16.1 How Do I Resolve "Declaration of xxxx must be compatible with xxxx problem"?

Such an error happens typically when versions are incompatible, given that dependencies for open-source software in the community are updated irregularly. The following gives an example error:

```
Declaration of Obs\Internal\Common\CheckoutStream::read($length) must be compatible with Psr\Http
\Message\StreamInterface::read(int $length)
```

This error says that **CheckoutStream::read($length)** does not declare the **int** type. A possible reason is that **psr/http-message** has a newer version, the number of which can be obtained from **composer.lock**. GitHub also shows that **psr/http-message** has been upgraded from version 1.1 to 2.0, with the **int** type declared in 2.0. To resolve this error, you can downgrade **psr/http-message** to version 1.1.

16.2 How Do I Get My Account ID and User ID?

Obtaining the Account ID and User ID

When calling APIs, you may need to specify the account ID (**DomainID**) and user ID (**UserID**) in some requests. You need to obtain them from the console in advance. To obtain them from the console, do as follows:

Step 1 Log in to the management console.

Step 2 Hover the mouse pointer over the username and choose **My Credentials** from the drop-down list.

On the **My Credentials** page, view the account ID and user ID.

----End

