Object Storage Service

BrowserJS 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 Before You Start

- To access a bucket using the OBS BrowserJS SDK, you must first **configure CORS for the bucket**.
- You can see **General Examples of ObsClient** to learn how to call OBS BrowserJS SDK APIs in a general manner.
- ObsClient supports API calling results returned via a callback function or the **Promise** object.
- 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.

2 SDK Download Links

SDK Source Code

- Latest version of OBS BrowserJS SDK source code: **Download**

Compatibility

- Recommended browsers:
  - Browsers that fully support HTML5
- Restrictions: Bucket creation, bucket listing, and CORS configuration are not supported.
- Interface functions: Not completely compatible with earlier versions (2.1.*x*). The following table describes the changes.

Interface Function	Description
ObsClient.setBucketAcl	In the request parameters, the type of the **Grants** field changes to **Array**, and the **Grants.Grant** field is deleted.
ObsClient.getBucketAcl	In the response parameters, the type of the **InterfaceResult.Grants** field changes to **Array**, and the **InterfaceResult.Grants.Grant** field is deleted.
ObsClient.setObjectAcl	In the request parameters, the type of the **Grants** field changes to **Array**, and the **Grants.Grant** field is deleted.
ObsClient.getObjectAcl	In the response parameters, the type of the **InterfaceResult.Grants** field changes to **Array**, and the **InterfaceResult.Grants.Grant** field is deleted.
ObsClient.setBucketLogging	In the request parameters, the type of the **LoggingEnabled.TargetGrants** field changes to **Array**, and the **LoggingEnabled.TargetGrants.Grant** field is deleted.

Interface Function	Description
ObsClient.getBucketLogging	In the response parameters, the type of the **InterfaceResult.LoggingEnabled.TargetGrants** field changes to **Array**, and the **InterfaceResult.LoggingEnabled.TargetGrants.Grant** field is deleted.
ObsClient.setBucketWebsite	In the request parameters, the type of the **RoutingRules** field changes to **Array**, and the **RoutingRules.RoutingRule** field is deleted.
ObsClient.getBucketWebsite	In the response parameters, the type of the **InterfaceResult.RoutingRules** field changes to **Array**, and the **InterfaceResult.RoutingRules.RoutingRule** field is deleted.
ObsClient.getBucketCors	In the response parameters, the **InterfaceResult.CorsRule** field is renamed as **InterfaceResult.CorsRules**.
ObsClient.setBucketTagging	In the request parameters, the type of the **TagSet** field changes to **Array**, and the **TagSet.Tag** field is deleted.
ObsClient.getBucketTagging	In the response parameters, the type of the **InterfaceResult.TagSet** field changes to **Array**, and the **InterfaceResult.TagSet.Tag** field is deleted.

3 Example Programs

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

You can obtain them from the OBS BrowserJS SDK.

After you download and decompress **eSDK_Storage_OBS_*<VersionId>*_BrowserJS.zip**, you can obtain example programs from **eSDK_Storage_OBS_*<VersionId>*_BrowserJS/examples**.

4 Quick Start

4.1 Creating an AK and SK

OBS uses access keys (AK and SK) for signature verification to ensure that only authorized accounts can access specified OBS resources. Detailed explanations are as follows:

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

The procedure is as follows:

1. Log in to OBS Console.
2. In the upper right corner of the page, click the username and choose **My Credentials**.
3. On the **My Credentials** page, click the **Access Keys** tab, and then click **Add Access Key** below the displayed access key list.
4. In the **Add Access Key** dialog box that is displayed, enter the password and its verification code.

   NOTE

   - If you have not bound an email address or mobile number, you need to enter only the password.
   - If you have bound an email address and a mobile number, you can select the verification either by email address or mobile number.
5. Click **OK**.
6. In the **Download Access Key** dialog box that is displayed, click **OK** to save the access keys to your browser's default download path.
7. Open the downloaded **credentials.csv** file to obtain the access keys (AK and SK).

NOTE

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

- 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.

4.2 Preparing a Development Environment

- Download Eclipse IDE for JavaScript and Web Developer from the **Eclipse's official website** and install it.
- Download and install a browser complying with the HTML5 standard, for example, Chrome 64.

4.3 Installing SDK

To install an OBS BrowserJS SDK, see **Table 4-1**.

Table 4-1 Methods of installing an OBS BrowserJS SDK

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 BrowserJS SDK of the latest version as an example:

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

Step 2 Decompress the development package to obtain folder **examples** (sample code), folder **dist** (SDK libraries), and file **README.txt** (feature description file of SDK versions).

Step 3 Import the SDK libraries to a browser's code.

```
<script src="./esdk-obs-browserjs-without-polyfill-x.x.x.min.js"></script>
```

----End

NOTICE

- The SDK is integrated with axios as the HTTP library that depends on the promise feature.
- The **dist** folder contains two versions of SDK libraries. If the browser you use does not support the promise feature, such as Internet Explorer 10 or 11, import library **esdk-obs-browserjs-x.x.x.min.js**.

4.4 Configuring CORS

To use the OBS BrowserJS SDK to access a bucket, you must configure cross-origin resource sharing (CORS) for the bucket. The following configuration is recommended for the CORS rule.

Parameter	Value	Description
Allowed Origin	*	Allows all request origins. **NOTE** You can also configure a domain name or IP address.
Allowed Method	PUT, GET, POST, DELETE, and HEAD	Allows all HTTP methods.
Allowed Header	*	Allows requests to carry any headers.

Parameter	Value	Description
Expose Header	• ETag • x-obs-request-id • x-obs-api • Content-Type • Content-Length • Cache-Control • Content-Disposition • Content-Encoding • Content-Language • Expires • x-obs-id-2 • x-reserved-indicator • x-obs-version-id • x-obs-copy-source-version-id • x-obs-storage-class • x-obs-delete-marker • x-obs-expiration • x-obs-website-redirect-location • x-obs-restore • x-obs-version • x-obs-object-type • x-obs-next-append-position	Allows the response to return the specified additional headers. **NOTICE** This parameter specifies the additional headers that a browser can expose to the client. Take **ETag** as an example. It is not a standard response header, so to obtain its value, you need to add it in this parameter for the browser to return the value in the response.

4.5 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Use the instance to access OBS.
```

NOTE

- For more information, see chapter "Initialization."
- For details about logging configuration, see **Configuring SDK Logging**.

4.6 Uploading an Object

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

The example code is as follows:

```
obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    Body : 'Hello OBS'
}, function (err, result) {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

NOTE

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

4.7 Downloading an Object

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

The example code is as follows:

```
obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname'
}, function (err, result) {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log(result.InterfaceResult.Content);
        }
    }
});
```

NOTE

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

4.8 Listing Objects

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

```
obsClient.listObjects({
    Bucket : 'bucketname'
}, function (err, result) {
```



```
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            for(var j in result.InterfaceResult.Contents){
                console.log('Contents[' + j +  ']:');
                console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                console.log('LastModified-->' + result.InterfaceResult.Contents[j]['LastModified']);
                console.log('Size-->' + result.InterfaceResult.Contents[j]['Size']);
            }
        }
    }
});
```

NOTE

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

4.9 Deleting an Object

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

The example code is as follows:

```
obsClient.deleteObject({
    Bucket : 'bucketname',
    Key : 'objectname'
}, function (err, result) {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

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**.

4.10 General Examples of ObsClient

Result Returned via a Callback Function

ObsClient returns the results by using a callback function that contains two parameters in sequence: the exception information parameter and the **SDK common result object** parameter. If the exception information parameter in the callback function is not null, an error occurs during the API calling. Otherwise, the API calling is complete. In such conditions, you need to obtain the HTTP status code from the **SDK common result object** parameter to check whether the operation is successful. Sample code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables
```

```
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Construct bucket request parameters.
var requestParam1 = {
        Bucket : 'bucketname'
        // Other fields
};

var callback1 = function (err, result){
        // Handle the API calling result.
};

// Call a bucket-related API, such as the API for obtaining the bucket location.
obsClient.getBucketLocation(requestParam1, callback1);

// Construct object request parameters.
var requestParam2 = {
        Bucket : 'bucketname',
        Key : 'objectname'
        // Other fields.
};

var callback2 = function (err, result){
        // Handle the API calling result.
};

// Call an object-related API, such as the API for downloading an object.
obsClient.getObject(requestParam2, callback2);
```

NOTE

For bucket-related APIs, the **Bucket** field contained in the request object specifies the bucket name. For object-related APIs, the **Bucket** and **Key** fields contained in the request object specify the bucket name and object name, respectively.

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Call an API to perform related operation, for example, uploading an object.
obsClient.putObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        Body : 'Hello OBS'
}, function (err, result) {
        // If the exception information is not null, the API calling is abnormal.
        if(err){
            console.log('Error-->' + err);
        }else{
            //If the exception information is null, the API calling is complete and the HTTP status code will be checked.
```

```
        if(result.CommonMsg.Status < 300){// Operation succeeded
            if(result.InterfaceResult){
                // Process the business logic after the operation is successful.
            }
        }else{// Obtain the exception details if the operation fails.
            console.log('Code-->' + result.CommonMsg.Code);
            console.log('Message-->' + result.CommonMsg.Message);
            console.log('HostId-->' + result.CommonMsg.HostId);
            console.log('RequestId-->' + result.CommonMsg.RequestId);
        }
    }

});
```

Result Returned via the Promise Object

ObsClient supports results returned via the **Promise** object. If no exception is caught by the **catch** method of the **Promise** object, the API calling is complete. In such conditions, you need to obtain the HTTP status code from the **SDK common result object** to check whether the operation is successful. Sample code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Construct bucket request parameters.
var requestParam1 = {
        Bucket : 'bucketname'
        // Other fields
};

// Call a bucket-related API, such as the API for obtaining the bucket location.
var promise1 = obsClient.getBucketLocation(requestParam1);
promise1.then((result) => {
   // Handle the API calling result.
}).catch((err)=>{
   // Rectify the fault.
});

// Construct object request parameters.
var requestParam2 = {
        Bucket : 'bucketname',
        Key : 'objectname'
        // Other fields.
};

// Call an object-related API, such as the API for downloading an object.
var promise2 = obsClient.getObject(requestParam2, callback2);
promise2.then((result) => {
    // Handle the API calling result.
}).catch((err)=>{
   // Rectify the fault.
});
```


> **NOTE**
>
> For bucket-related APIs, the **Bucket** field contained in the request object specifies the bucket name. For object-related APIs, the **Bucket** and **Key** fields contained in the request object specify the bucket name and object name, respectively.

Sample code:

```
// Import the OBS library.
var ObsClient = require('./lib/obs');

// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Call an API to perform related operation, for example, uploading an object.
obsClient.putObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        Body : 'Hello OBS'
}).then((result) => {
    // If no exception occurs and the API call is complete, check the HTTP status code.
    if(result.CommonMsg.Status < 300){// Operation succeeded
        if(result.InterfaceResult){
                        // Process the business logic after the operation is successful.
        }
            }else{// Obtain the exception details if the operation fails.
        console.log('Code-->' + result.CommonMsg.Code);
        console.log('Message-->' + result.CommonMsg.Message);
        console.log('HostId-->' + result.CommonMsg.HostId);
        console.log('RequestId-->' + result.CommonMsg.RequestId);
    }
}).catch((err) => {
    // An exception occurred after the API is called.
    console.error('Error-->' + err);
});
```

4.11 Pre-defined Constants

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

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

For details, see the *OBS BrowserJS SDK API Reference*.

5 Initialization

5.1 Configuring CORS for a Bucket

OBS provides distributed storage service over HTTP/HTTPS, and stores data in buckets. However, browsers do not allow the cross-origin AJAX requests by default. For this reason, configure CORS before using the OBS BrowserJS SDK to access buckets. You can configure CORS on OBS Console, on OBS Browser, or using other OBS SDKs (excluding the OBS BrowserJS SDK). The following configuration is recommended for the CORS rule.

Parameter	Value	Description
Allowed Origin	*	Allows all request origins. **NOTE** You can also configure a domain name or IP address.
Allowed Method	PUT, GET, POST, DELETE, and HEAD	Allows all HTTP methods.
Allowed Header	*	Allows requests to carry any headers.

Parameter	Value	Description
Expose Header	• ETag • x-obs-request-id • x-obs-api • Content-Type • Content-Length • Cache-Control • Content-Disposition • Content-Encoding • Content-Language • Expires • x-obs-id-2 • x-reserved-indicator • x-obs-version-id • x-obs-copy-source-version-id • x-obs-storage-class • x-obs-delete-marker • x-obs-expiration • x-obs-website-redirect-location • x-obs-restore • x-obs-version • x-obs-object-type • x-obs-next-append-position	Allows the response to return the specified additional headers. **NOTICE** This parameter specifies the additional headers that a browser can expose to the client. Take **ETag** as an example. It is not a standard response header, so to obtain its value, you need to add it in this parameter for the browser to return the value in the response.

Configuring CORS on OBS Console

Step 1 Log in to OBS Console. In the bucket list, click the bucket you want. The bucket **Overview** page is displayed.

Step 2 In the **Basic Configurations** area, click **CORS Rules**. The **CORS Rules** page is displayed.

Step 3 Click **Create**. The **Create CORS Rule** dialog box is displayed. Set the parameters according to the preceding table. See the following figure.

Step 4 Click **OK**. On the **CORS Rules** page, view the configured rule.

----End

NOTE

The CORS configuration of a bucket takes effect in two minutes. You can access the bucket using the OBS BrowserJS SDK only after the configuration takes effect.

Configuring CORS Using the OBS Java SDK

You can call **ObsClient.setBucketCors** of the OBS Java SDK to configure the CORS rule for a bucket. The sample code is as follows:

```
String endPoint = "https://your-endpoint"
String ak = System.getenv("ACCESS_KEY_ID");
String sk = System.getenv("SECRET_ACCESS_KEY_ID");
// Create an ObsClient instance.
ObsClient obsClient = new ObsClient(ak, sk, endPoint);

BucketCors cors = new BucketCors();

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

// Specify the origin of the cross-origin request.
ArrayList<String> allowedOrigin = new ArrayList<String>();
allowedOrigin.add( "*");
rule.setAllowedOrigin(allowedOrigin);

// Specify the methods of the cross-origin request.
ArrayList<String> allowedMethod = new ArrayList<String>();
allowedMethod.add("GET");
allowedMethod.add("POST");
allowedMethod.add("PUT");
allowedMethod.add("DELETE");
allowedMethod.add("HEAD");
rule.setAllowedMethod(allowedMethod);

// Specify the headers of the cross-origin request.
ArrayList<String> allowedHeader = new ArrayList<String>();
allowedHeader.add("*");
rule.setAllowedHeader(allowedHeader);

// Specify the additional headers in the response to the cross-origin request.
ArrayList<String> exposeHeader = new ArrayList<String>();
exposeHeader.add("ETag");
exposeHeader.add("Content-Type");
exposeHeader.add("Content-Length");
exposeHeader.add("Cache-Control");
exposeHeader.add("Content-Disposition");
exposeHeader.add("Content-Encoding");
exposeHeader.add("Content-Language");
exposeHeader.add("Expires");
exposeHeader.add("x-obs-request-id");
exposeHeader.add("x-obs-id-2");
exposeHeader.add("x-reserved-indicator");
exposeHeader.add("x-obs-api");
exposeHeader.add("x-obs-version-id");
exposeHeader.add("x-obs-copy-source-version-id");
exposeHeader.add("x-obs-storage-class");
exposeHeader.add("x-obs-delete-marker");
exposeHeader.add("x-obs-expiration");
exposeHeader.add("x-obs-website-redirect-location");
exposeHeader.add("x-obs-restore");
exposeHeader.add("x-obs-version");
exposeHeader.add("x-obs-object-type");
exposeHeader.add("x-obs-next-append-position");
rule.setExposeHeader(exposeHeader);

rule.setMaxAgeSecond(100);
```



```
rules.add(rule);
cors.setRules(rules);

try
{
    obsClient.setBucketCors("bucketname", cors);
catch (ObsException e)
{
    System.out.println("HTTP Code: " + e.getResponseCode());
    System.out.println("Error Code:" + e.getErrorCode());
    System.out.println("Error Message: " + e.getErrorMessage());

    System.out.println("Request ID:" + e.getErrorRequestId());
    System.out.println("Host ID:" + e.getErrorHostId());
}
}
```

Configure CORS Using OBS Python SDK

You can call **ObsClient.setBucketCors** of the OBS Python SDK to configure the CORS of a bucket. The sample code is as follows:

```
# Import the module.
from obs import 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.
ak = os.getenv("AccessKeyID")
sk = os.getenv("SecretAccessKey")
# (Optional) If you use a temporary AK/SK pair and a security token to access OBS, obtain them from environment variables.
security_token = os.getenv("SecurityToken")
# Set server to the endpoint of the region where the bucket is located.
server = "https://your-endpoint"

# Create an obsClient instance.
# If you use a temporary AK/SK pair and a security token to access OBS, you must specify security_token when creating an instance.
obsClient = ObsClient(access_key_id=ak, secret_access_key=sk, server=server)

from obs import CorsRule

# Specify the origin of the cross-origin request.
allowedOrigin = ['*']
# Specify the methods of the cross-origin request.
allowedMethod = ['PUT', 'POST', 'GET', 'DELETE', 'HEAD']
# Specify the headers of the cross-origin request.
allowedHeader = ['*']

# Specify the additional headers in the response to the cross-origin request.
exposeHeader = ['ETag', 'Content-Type', 'Content-Length', 'Cache-Control', 'Content-Disposition', 'Content-Encoding', 'Content-Language', 'Expires', 'x-obs-request-id', 'x-obs-id-2', 'x-reserved-indicator', 'x-obs-api', 'x-obs-version-id', 'x-obs-copy-source-version-id', 'x-obs-storage-class', 'x-obs-delete-marker', 'x-obs-expiration', 'x-obs-website-redirect-location', 'x-obs-restore', 'x-obs-version', 'x-obs-object-type', 'x-obs-next-append-position']

maxAgeSecond = 100
cors = CorsRule(id='rule1', allowedMethod=allowedMethod,
                allowedOrigin=allowedOrigin, allowedHeader=allowedHeader,
                maxAgeSecond=maxAgeSecond, exposeHeader=exposeHeader)

resp = obsClient.setBucketCors('bucketname', corsList=[cors])
if resp.status < 300:
    print('requestId:', resp.requestId)
else:
    print('errorCode:', resp.errorCode)
    print('errorMessage:', resp.errorMessage)
```


NOTE

- Except the OBS BrowserJS SDK, all the other OBS SDKs can be used to configure CORS rules for buckets.

5.2 Configuring the AK and SK

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

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

For details, see **Creating an AK and SK**.

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

- **Creating an Instance of ObsClient**
- **Configuring an Instance of ObsClient**
- **Configuring SDK Logging**

5.3 Creating an Instance of ObsClient

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

Direction Creation

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

```
// AMD is not introduced. Use the constructor to create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Use the instance to access OBS.
```

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

```
// AMD is not introduced. Use the constructor to create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID, SecretAccessKey, and SecurityToken.
    // The front-end code does not have the process environment variable, so you need to use a
```



```
module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    security_token: process.env.SecurityToken,
    server: 'https://your-endpoint'
});

// Use the instance to access OBS.
```

Creation According to AMD

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

```
// AMD is introduced. Use the injected constructor to create an instance of ObsClient.
var obsClient;
define(['ObsClient'], function(ObsClient){
    obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
        // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
        // Obtain an AK/SK pair on the management console.
        access_key_id: process.env.AccessKeyID,
        secret_access_key: process.env.SecretAccessKey,
        server: 'https://your-endpoint'
    });

    // Use the instance to access OBS.
});
```

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

```
// AMD is introduced. Use the injected constructor to create an instance of ObsClient.
var obsClient;
define(['ObsClient'], function(ObsClient){
    obsClient = new ObsClient({
        // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID, SecretAccessKey, and SecurityToken.
        // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
        // Obtain an AK/SK pair on the management console.
        access_key_id: process.env.AccessKeyID,
        secret_access_key: process.env.SecretAccessKey,
        security_token: process.env.SecurityToken,
        server: 'https://your-endpoint'
    });

    // Use the instance to access OBS.
});
```

5.4 Configuring an Instance of ObsClient

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

Parameter	Description	Recommended Value
access_key_id	AK	N/A

Parameter	Description	Recommended Value
secret_access_key	SK	N/A
server	Endpoint for accessing OBS, which contains the protocol type, domain name (or IP address), and port number. For example, https://your-endpoint:443. For security purposes, you are advised to use HTTPS.	N/A
timeout	The total timeout period (in seconds) of an HTTP/HTTPS request. The default value is **300**.	[10,300]
is_cname	Whether to use self-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
useRawXhr	Whether to use the native XHR to send Ajax requests. The default value is **false**.	N/A

NOTE

- Parameters whose recommended value is **N/A** need to be set according to the actual conditions.
- If the network is unstable or the size of the file to be uploaded is large, you are advised to set a larger value for **timeout**.

5.5 Configuring SDK Logging

You can call **ObsClient.initLog** to enable bucket logging. Sample code is as follows:

```
obsClient.initLog({
    level:'warn', // Set the log level.
});
```

NOTE

- Logs printed by the SDK will be displayed on the Console within the developer tools of a browser.
- The logging function is disabled by default. You need to enable it manually.
- For details about SDK logs, see **Log Analysis**.

6 Fault Locating

6.1 Methods

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

Step 1 Make sure that the OBS BrowserJS SDK is the latest version. **Download the latest version**.

Step 2 Make sure that the program code of the OBS BrowserJS SDK complies with **General Examples of ObsClient**. All ObsClient APIs are processed with exception handling. The following is an example code of uploading an object:

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.putObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        Body : 'Hello OBS'
}, function (err, result) {
        if(err){
            // If there are any error messages displayed, there is something wrong with the API call. The common cause is a network exception.
            console.error('Error-->' + err);
        }else{
            // If there are no error messages displayed, the API call is complete. Then, check the HTTP status code using the SDK common result.
            if(result.CommonMsg.Status < 300){// If the HTTP status code is less than 300, and the API call is successful.
                if(result.InterfaceResult){
                    // Process the business logic after the call is successful.
                    // Optional: After the API is successfully called, record the HTTP status code and request ID returned by the server.
                    console.log('Status-->' + result.CommonMsg.Status);
```



```
                console.log('RequestId-->' + result.CommonMsg.RequestId);
            }
        }else{
            // Recommended: If the call fails, record the HTTP status code, server-side error code, and
request ID returned by the server.
            console.log('Status-->' + result.CommonMsg.Status);
            console.log('Code-->' + result.CommonMsg.Code); ;
            console.log('RequestId-->' + result.CommonMsg.RequestId);
        }
    }
});
```

NOTE

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

Step 3 If there are any error messages, check the network connection between the client and the OBS server first. If the connection is good, provide the error messages to the OBS client O&M team for locating the cause.

Step 4 If an ObsClient API call fails, obtain the **HTTP status code** and **OBS server-side error code** from the SDK common result objects, and compare them to locate the cause.

Step 5 If the cause cannot be found in step 4, obtain the request ID returned by the OBS server from the SDK common result objects and contact the OBS server O&M team to locate the cause.

Step 6 If the request ID cannot be obtained from the SDK common result objects, contact the OBS client O&M team to locate the failure cause.

----End

6.2 Notable Issues

SignatureDoesNotMatch

```
Status-->403
Code-->SignatureDoesNotMatch
```

This error occurs if the SK input into ObsClient initialization is incorrect. Solution: Make sure that the SK is correct.

MethodNotAllowed

```
Status-->405
Code-->MethodNotAllowed
```

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

Network Error

```
Error: Network Error
```

Possible causes are as follows:

1. The endpoint input into ObsClient initialization is incorrect. Solution: Verify to make sure that the endpoint is correct.

2. The network between the OBS client and OBS server is abnormal. Solution: Check the health status of the network.
3. The OBS domain name resolved by DNS is inaccessible. Solution: Contact the OBS O&M team.
4. The SDK depends on the compatibility of the underlying library Axios. Solution: **Perform a browser-based upload** or contact the OBS O&M team.

Request Timeout

```
timeout of xxx exceeded
```

Possible causes are as follows:

1. The network latency between the OBS client and OBS server is too long. Solution: Check the health status of the network.
2. The network connection between the OBS client and OBS server is abnormal. Solution: Check the network status.

Cross-Origin Requests are Blocked

```
Access to XMLHttpRequest at 'xxx' from origin 'xxx' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.
```

This error occurs because the bucket does not have the CORS rule or the CORS rule is invalid. Solution: Reconfigure the bucket CORS by referring to **Configuring the Bucket CORS**.

A Loading Error Occurs After the SDK Is Integrated into the Internet Explorer

```
SCRIPT5009: 'Promise' is not defined.
```

The cause is that the Internet Explorer does not support the Promise object (ES6). The solutions are as follows:

1. Use **esdk-obs-browserjs-x.x.x.min.js** as the SDK library file instead of **esdk-obs-browserjs-without-polyfill-x.x.x.min.js** in the program.
2. Introduce a third-party library to the Internet Explorer to supplement ES6. For example, introduce the **babel-polyfill** library.
3. Use browsers that support ES6, such as Chrome or Firefox.

Failed to Obtain the ETag Value after Upload

After a file is successfully uploaded by calling **ObsClient.putObject** or **ObsClient.uploadPart**, the returned result does not contain the ETag value. The possible causes are as follows:

1. The ETag header was not included in **ExposeHeader** in the CORS configuration of the bucket. Solution: Configure CORS for the bucket by referring to **Configuring CORS for a Bucket**.
2. The ETag header was included in **ExposeHeader** in the CORS configuration of the bucket but was just shielded in the result returned by the browser. This problem usually occurs on the browser of an earlier version. Solution: Upgrade the browser to a new version that supports HTML5.

ObsClient Is Not Defined

```
Uncaught ReferenceError: ObsClient is not defined
```

Possible causes are as follows:

1. The SDK is not correctly introduced to the program. Solution: Check whether the **esdk-obs-browserjs-x.x.x.min.js** or **esdk-obs-browserjs-without-polyfill-x.x.x.min.js** file is correctly introduced by checking the method of introducing the SDK.
2. The AMD modular components, such as **require.js**, are introduced to the program. Solution: **Create an instance of ObsClient according to AMD**.
3. A component introduced in the program conflicts with the SDK dependent library (this scenario seldom occurs). Solution: Contact the OBS O&M team.

Unable to Upload Files Using a Browser that Does Not Support window.File

```
Error: source file must be an instance of window.File or window.Blob
```

The SDK depends on **window.File** provided by HTML5 to upload files. For browsers that do not support **window.File**, such as IE8 and IE9, files cannot be uploaded by calling **ObsClient.putObject** or **ObsClient.uploadFile**. Solution: Perform a browser-based upload. The procedure is as follows:

Step 1 Check whether the browser supports **window.File**. The code example is as follows:

```
function getBrowserInfo() {
    var agent = navigator.userAgent.toLowerCase();
    var regStr_ie = /msie [\d.]+;/gi;
    var regStr_ff = /firefox\/[\d.]+/gi
    var regStr_chrome = /chrome\/[\d.]+/gi;
    var regStr_saf = /safari\/[\d.]+/gi;
    var isIE = agent.indexOf('compatible') > -1 && agent.indexOf('msie' > -1);
    var isEdge = agent.indexOf('edge') > -1 && !isIE;
    var isIE11 = agent.indexOf('trident') > -1 && agent.indexOf('rv:11.0') > -1;
    if (isIE) {
        var reIE = new RegExp('msie (\\d+\\.\\d+);');
        reIE.test(agent);
        var fIEVersion = parseFloat(RegExp['$1']);
        if (fIEVersion == 7) {
            return 'IE/7';
        } else if (fIEVersion == 8) {
            return 'IE/8';
        } else if (fIEVersion == 9) {
            return 'IE/9';
        } else if (fIEVersion == 10) {
            return 'IE/10';
        }
    }
    // isIE end
    if (isIE11) {
        return 'IE/11';
    }
    // Firefox
    if (agent.indexOf('firefox') > 0) {
        return agent.match(regStr_ff);
    }
    // Safari
    if (agent.indexOf('safari') > 0 && agent.indexOf('chrome') < 0) {
        return agent.match(regStr_saf);
    }
    // Chrome
    if (agent.indexOf('chrome') > 0) {
        return agent.match(regStr_chrome);
    }
```



```
    return '';
}

var browserInfo = getBrowserInfo();
// Check whether the browser supports window.File.
var isSupportFileApi = browserInfo !== 'IE/7' && browserInfo !== 'IE/8' && browserInfo !== 'IE/9' &&
window.File;
```

Step 2 Select a proper upload method based on the result in step 1. The code example is as follows:

```
function postObject(){
  // Use JS code to submit a form for the browser-based upload.
}
if(isSupportFileApi){
    // Upload files in browser-based mode.
    return postObject();
}
// Create an ObsClient instance.
var obsClient = new ObsClient({
   // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
   // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
   // Obtain an AK/SK pair on the management console.
   access_key_id: process.env.AccessKeyID,
   secret_access_key: process.env.SecretAccessKey,
   server: 'https://your-endpoint
});
// Use the resumable upload API of the SDK to upload files.
obsClient.uploadFile({
  // Transfer the request parameter.
}, function (err, result) {
  // Process the callback function.
});
```

----End

NOTE

The size of a file for browser-based upload cannot exceed 5 GB.

Undefinition Caused by the Introduction of the CommonJS Specification

```
Uncaught (in promise) TypeError: Cannot read property 'CancelToken' of undefined
```

The cause is that a modular component of the CommonJS specification is introduced to the program, such as webpack. Solution: Upgrade the SDK to 3.19.5 or a later version.

Undefinition Caused by the Introduction of Mock.js

```
Uncaught TypeError: request.upload.addEventListener in not a function
```

The cause is that the Mock.js component is used to stub XHR in the program. The solutions are as follows:

1. Workaround: Disable the progress bar function when the SDK is used for upload, download, and resumable upload.
2. Replace: Replace the Mock.js component with a component that can simulate all XHR interfaces.

3. Extend: Extend the Mock.js component by supplementing the Mock.js interfaces that do not support XHR.

7 Bucket Management

7.1 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Obtain the bucket metadata.
obsClient.getBucketMetadata({
        Bucket: 'bucketname'
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                console.log('StorageClass-->' + result.InterfaceResult.StorageClass);
                console.log('RequestId-->' + result.InterfaceResult.RequestId);
            }
        }
});
```

NOTE

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

7.2 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.headBucket({
        Bucket: 'bucketname'
}, function(err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            if(result.CommonMsg.Status < 300){
                console.log('Bucket exists');
            }else if(result.CommonMsg.Status === 404){
                console.log('Bucket does not exist');
            }
        }
});
```

NOTE

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

7.3 Deleting a Bucket

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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Delete a bucket.
obsClient.deleteBucket({
        Bucket: 'bucketname'
}, function(err, result) {
        if(err){
```

```
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

NOTE

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

7.4 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. Call ObsClient.setBucketAcl to specify a pre-defined ACL.
2. Call ObsClient.setBucketAcl to specify a user-defined ACL.

OBS supports five types of bucket or object ACL configuration items, as listed in the following table:

Permission	Description	Value in OBS BrowserJS 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.enums.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.enums.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.enums.PermissionReadAcp

Permission	Description	Value in OBS BrowserJS SDK
WRITE_ACP	A grantee with this permission can update the ACL of a bucket or object. A bucket or object owner has this permission permanently. A grantee with this permission can modify the access control policy and thus the grantee obtains full access permissions.	ObsClient.enums.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.enums.PermissionFullControl

OBS pre-defined ACLs are classified into five types, as shown in the following table:

Policy	Description	Value in OBS BrowserJS 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.enums.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.enums.AclPublicRead

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

Setting a Pre-defined ACL for a Bucket

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
```



```
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Set the bucket ACL to a pre-defined ACL.
obsClient.setBucketAcl({
        Bucket: 'bucketname',
        // Set the bucket ACL to private read and write.
        ACL: obsClient.enums.AclPrivate
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

Setting a User-defined Bucket ACL

The following code shows how to set a user-defined ACL for a bucket:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Directly set the bucket ACL.
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.enums.PermissionFullControl},
            // Grant the READ permission to all users.
            { Grantee : {Type : 'Group',URI : obsClient.enums.GroupAuthenticatedUsers}, Permission : obsClient.enums.AclPublicRead}
        ]
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

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.enums.GroupAllUsers

Obtaining a Bucket's ACL

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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getBucketAcl({
        Bucket: 'bucketname',
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                console.log('RequestId-->' + result.InterfaceResult.RequestId);
                console.log('Owner[ID]-->' + result.InterfaceResult.Owner.ID);
                console.log('Grants:');
                for(var i in result.InterfaceResult.Grants){
                    console.log('Grant[' + i + ']:');
                    console.log('Grantee[ID]-->' + result.InterfaceResult.Grants[i]['Grantee']['ID']);
                    console.log('Grantee[URI]-->' + result.InterfaceResult.Grants[i]['Grantee']['URI']);
                    console.log('Permission-->' + result.InterfaceResult.Grants[i]['Permission']);
                }
            }
        }
});
```

7.5 Management 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 bucket policies. Sample code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});
// Bucket name
```



```
const bucketName = 'bucketname';
// Bucket policy
const policy =  "{\"Statement\":[{\"Principal\":\"*\",\"Effect\":\"Allow\",\"Action\":\"ListBucket\",\"Resource
\":\""+bucketName+"\"}]}";
//Set a bucket policy.
obsClient.setBucketPolicy({
    Bucket: bucketName,
    Policy: policy
}, function(err, result) {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

NOTE

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

Obtaining a Bucket Policy

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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
the configuration file or environment variables. In this example, the AK/SK are stored in environment
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Obtain the bucket policy.
obsClient.getBucketPolicy({
    Bucket: 'bucketname',
}, function(err, result) {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('Policy-->' + result.InterfaceResult.Policy);
        }
    }
});
```

Deleting a Bucket Policy

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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
the configuration file or environment variables. In this example, the AK/SK are stored in environment
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
```



```
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Delete a bucket policy.
obsClient.deleteBucketPolicy({
        Bucket: 'bucketname'
}, function(err, result) {
        if(err){
                console.error('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

7.6 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getBucketLocation({
        Bucket: 'bucketname',
}, function (err, result) {
        if(err){
                console.error('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                        console.log('Location-->' + result.InterfaceResult.Location);
                }
        }
});
```

NOTE

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

7.7 Obtaining Storage Information About a Bucket

The storage information about a bucket includes the bucket size 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getBucketStorageInfo({
        Bucket: 'bucketname',
},function (err, result) {
        if(err){
                console.error('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                        console.log('Size-->' + result.InterfaceResult.Size);
                        console.log('ObjectNumber-->' + result.InterfaceResult.ObjectNumber);
                }
        }
});
```

NOTE

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

7.8 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

//Set the bucket quota to 100 MB.
obsClient.setBucketQuota({
        Bucket: 'bucketname',
        StorageQuota: 1024 * 1024 * 100
}, function (err, result) {
        if(err){
                console.error('Error-->' + err);
        }else{
```

```
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getBucketQuota({
      Bucket : 'bucketname'
}, function (err, result) {
      if(err){
            console.error('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                  console.log('StorageQuota-->' + result.InterfaceResult.StorageQuota);
            }
      }
});
```

7.9 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 BrowserJS 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.enums.StorageClassStandard

Storage Class	Description	Value in OBS BrowserJS SDK
OBS Warm	Is applicable to storing semi-frequently accessed (less than 12 times a year) data requiring quick response.	ObsClient.enums.StorageClassWarm
OBS Cold	Is applicable to archiving rarely-accessed (once a year) data.	ObsClient.enums.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.enums.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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.setBucketStoragePolicy({
        Bucket: 'bucketname',
        StorageClass: obsClient.enums.StorageClassWarm
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
```



```
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getBucketStoragePolicy({
        Bucket: 'bucketname'
}, function (err, result) {
        if(err){
                console.error('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                        console.log('StorageClass-->' + result.InterfaceResult.StorageClass);
                }
        }
});
```

8 Object Upload

8.1 Object Upload Overview

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

- **Performing a Text-Based Upload**
- **Performing a File-Based Upload**
- **Performing a Multipart Upload**
- **Performing an Appendable 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, file-based upload is 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.

8.2 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment
```



```
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.putObject({
        Bucket: 'bucketname',
        // An object name is a complete path that does not contain the bucket name.
        Key: 'objectname',
        // Content of the object to be uploaded
        Body: 'Hello OBS'
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

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

8.3 Performing a File-Based Upload

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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
the configuration file or environment variables. In this example, the AK/SK are stored in environment
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.putObject({
        Bucket: 'bucketname',
        Key: 'objectname',
        SourceFile: document.getElementById('input-file').files[0]
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

- Use the **SourceFile** parameter to specify the to-be-uploaded file (a File or Blob object). For example, on an HTML page, use an input tag whose type is **file** to specify the to-be-uploaded file: **<input type="file" id="input-file"/>**.
- The **SourceFile** parameter and the **Body** parameter cannot be used together.
- The content to be uploaded cannot exceed 5 GB.

NOTICE

- The browser you use must support the **window.File** feature. Otherwise, files cannot be uploaded properly.
- For details about how to solve the problem, see **Unable to Upload Files Using a Browser that Does Not Support window.File**.

8.4 Obtaining Upload Progresses

You can set the callback function to obtain upload progress.

This example uploads a file to bucket **bucketname** as object **objectname** and uses **ProgressCallback** to monitor the upload progress.

The example code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

var callback = function(transferredAmount, totalAmount, totalSeconds){
    // Obtain the average upload rate (KB/s).
    console.log(transferredAmount * 1.0 / totalSeconds / 1024);
    // Obtain the upload progress in percentage.
    console.log(transferredAmount * 100.0 / totalAmount);
};

obsClient.putObject({
        Bucket: 'bucketname',
        Key: 'objectname',
        SourceFile: document.getElementById('input-file').files[0],
        ProgressCallback: callback
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```


NOTE

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

8.5 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.

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.putObject({
        Bucket: 'bucketname',
        Key: 'parent_directory/'
}, function (err, result) {
        if(err){
                console.error('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
        }
});
// Create an object in the folder.
obsClient.putObject({
        Bucket : 'bucketname',
        Key : 'parent_directory/objectname'
}, function (err, result) {
        if(err){
                console.error('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

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.

8.6 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 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 file length, the object will be truncated.	Actual length of the 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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.putObject({
        Bucket: 'bucketname',
```



```
    Key: 'objectname',
    SourceFile: document.getElementById('input-file').files[0],
    ContentLength: 1024 * 1024 // 1 MB
}, function (err, result) {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

NOTE

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

Setting the MIME Type for an Object

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Upload an image.
obsClient.putObject({
    Bucket: 'bucketname',
    Key: 'objectname.jpg',
    SourceFile: document.getElementById('input-file').files[0],
    ContentType: 'image/jpeg'
}, function (err, result) {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

NOTE

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

Setting the MD5 Value for an Object

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});
```



```
obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    SourceFile : document.getElementById('input-file').files[0],
    ContentMD5 : 'your md5 which should be encoded by base64'
}, function (err, result) {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    SourceFile : document.getElementById('input-file').files[0],
    // Set the storage class to Cold.
    StorageClass : ObsClient.enums.StorageClassCold
}, function (err, result) {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

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.
- Before downloading a Cold object, you must restore it.

Customizing Metadata for an Object

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.putObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        SourceFile : document.getElementById('input-file').files[0],
        Metadata : {'property1':'property-value1', 'property2' : 'property-value2'},
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

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

8.7 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 connection to the OBS server breaks often.
- Sizes of files to be uploaded are uncertain.

Multipart upload consists of three phases:

Step 1 Initiate 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

NOTE

You can also call the **API for resumable upload** (encapsulation and enhancement of multipart upload) provided by the SDK to implement multipart upload.

Initiating a Multipart Upload

Before using a multipart upload, you need to first 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 initiate a multipart upload.

```
// Create an instance of ObsClient.var obsClient = new ObsClient({    // Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK before storing them in the configuration file or environment variables. In this example, the AK and SK are stored in environment variables. Before running the code in this example, configure environment variables AccessKeyID and SecretAccessKey.    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,    secret_access_key: process.env.SecretAccessKey,    server: 'https://your-endpoint'});obsClient.initiateMultipartUpload({       Bucket : 'bucketname',       Key : 'objectname',       ContentType : 'text/plain',       Metadata : {'property' : 'property-value'}}, function (err, result) {       if(err){             console.error('Error-->' + err);       }else{             console.log('Status-->' + result.CommonMsg.Status);             if(result.CommonMsg.Status < 300 && result.InterfaceResult){                   console.log('UploadId-->' + result.InterfaceResult.UploadId);             }       }});
```

NOTE

- When initiating a multipart upload, you can use the **ContentType** and **Metadata** parameters to respectively set the MIME type and custom metadata of an object.
- After the API for initiating a multipart upload is successfully called, an upload ID will be returned. This ID will be used in follow-up operations.

Uploading a Part

After initiating a multipart upload, you can specify the object name and upload ID to upload a part. Each upload part has a part number (ranging from **1** to **10000**). For parts with the same upload ID, their part numbers are unique and identify their relative location in the object. If you use the same part number to upload two parts, the latter one uploaded will overwrite the former one. The last part uploaded can be up to 5 GB in size, and **the size of each of the other parts is in the range of 100 KB to 5 GB**. Parts can be uploaded in random order, or even through different processes or machines. OBS will combine them into the final object based on their part numbers.

You can call **ObsClient.uploadPart** to upload parts.

```
// Create an instance of ObsClient.var obsClient = new ObsClient({    // Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK before storing them in the configuration file or environment variables. In this example, the AK and SK are stored in environment variables. Before running the code in this example, configure environment variables AccessKeyID and SecretAccessKey.    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,    secret_access_key: process.env.SecretAccessKey,    server: 'https://your-endpoint'});const bucketname = 'examplebucket';const objectname = 'exampleobject';const PartSize = 5 * 1024 * 1024;const Uploadid = 'upload id from initiateMultipartUpload';const file = document.getElementById('input-file').files[0];const lastPartSize = file.size % PartSize;// Number of partsconst count = Math.ceil(file.size / PartSize);// Upload part n.const uploadPart = (n) => {    obsClient.uploadPart({       Bucket: bucketname,       Key: objectname,       // Set the part number,
```

```
which ranges from 1 to 10000.        PartNumber: n,        // Set the upload ID.        UploadId,        // Specify
the large file to be uploaded.        SourceFile: file,        // Set the part size.        PartSize: count === n ?
lastPartSize : PartSize,        // Set the start offset.        Offset: (n-1) * PartSize    }, function (err, result)
{        if(err){                console.log('Error-->' + err);        }else{                console.log('Status-->' +
result.CommonMsg.Status);                if(result.CommonMsg.Status < 300 && result.InterfaceResult)
{                        console.log('ETag-->' + result.InterfaceResult.ETag);                }        }    });}// Upload part
1.uploadPart(1);
```

CAUTION

If the ETag value obtained is **undefined**, you need to configure a CORS rule and add the ETag to the additional header. For details, see **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 the file.
- **SourceFile** must indicate a File or Blob object. For example, on an HTML page, use an input tag whose type is **file** to specify the to-be-uploaded file: **<input type="file" id="input-file"/>**.
- 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.

```
// Create an instance of ObsClient.var obsClient = new ObsClient({    // Hard-coded or plaintext AK and SK
are risky. For security purposes, encrypt your AK and SK before storing them in the configuration file or
environment variables. In this example, the AK and SK are stored in environment variables. Before running
the code in this example, configure environment variables AccessKeyID and SecretAccessKey.    // The
front-end code does not have the process environment variable, so you need to use a module bundler like
webpack to define the process variable.    // Obtain an AK/SK pair on the management console.
access_key_id: process.env.AccessKeyID,    secret_access_key: process.env.SecretAccessKey,    server: 'https://
your-endpoint'});obsClient.completeMultipartUpload({        Bucket:'bucketname',
Key:'objectname',        // Set the upload ID.        UploadId:'upload id from initiateMultipartUpload',        Parts:
[{'PartNumber':1,'ETag':'etag value from uploadPart'}]}, function (err, result) {        if(err)
{                console.log('Error-->' + err);        }else{                console.log('Status-->' +
result.CommonMsg.Status);        }});
```


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 are generated in the bucket. These parts occupy your storage space. You can cancel the multi-part uploading task to delete unnecessary parts, thereby saving the storage space.

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

```
// Create an instance of ObsClient.var obsClient = new ObsClient({    // Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK before storing them in the configuration file or environment variables. In this example, the AK and SK are stored in environment variables. Before running the code in this example, configure environment variables AccessKeyID and SecretAccessKey.    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,    secret_access_key: process.env.SecretAccessKey,    server: 'https://your-endpoint'});obsClient.abortMultipartUpload({       Bucket:'bucketname',       Key:'objectname',       // Set the upload ID.       UploadId:'upload id from initiateMultipartUpload',}, function (err, result) {       if(err){              console.log('Error-->' + err);       }else{              console.log('Status-->' + result.CommonMsg.Status);       }});
```

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

```
// Create an instance of ObsClient.var obsClient = new ObsClient({    // Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK before storing them in the configuration file or environment variables. In this example, the AK and SK are stored in environment variables. Before running the code in this example, configure environment variables AccessKeyID and SecretAccessKey.    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,    secret_access_key: process.env.SecretAccessKey,    server: 'https://your-endpoint'});// List uploaded parts. uploadId is obtained from initiateMultipartUpload.obsClient.listParts({       Bucket : 'bucketname',       Key: 'objectname',       UploadId : 'upload id from initiateMultipartUpload'}, function (err, result) {       if(err){            console.log('Error-->' + err);       }else{            console.log('Status-->' + result.CommonMsg.Status);            if(result.CommonMsg.Status < 300 && result.InterfaceResult){                for(var i in result.InterfaceResult.Parts){                    console.log('Part['+ i +']:');                    // Part number, specified upon uploading                    console.log('PartNumber-->' + result.InterfaceResult.Parts[i]['PartNumber']);                    // Time when the part was last uploaded                    console.log('LastModified-->' + result.InterfaceResult.Parts[i]['LastModified']);                    // Part ETag                    console.log('ETag-->' + result.InterfaceResult.Parts[i]['ETag']);                    // Part size                    console.log('Size-->' + result.InterfaceResult.Parts[i]['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, **InterfaceResult.IsTruncated** in the response is **true**, indicating not all parts were listed. In such case, you can use **InterfaceResult.NextPartNumberMarker** to obtain the start position for the 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

The following sample code lists more than 1,000 parts:

```
// Create an instance of ObsClient.var obsClient = new ObsClient({    // Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK before storing them in the configuration file or environment variables. In this example, the AK and SK are stored in environment variables. Before running the code in this example, configure environment variables AccessKeyID and SecretAccessKey.    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,    secret_access_key: process.env.SecretAccessKey,    server: 'https://your-endpoint'});var listAll = function (partNumberMarker) {       // List uploaded parts. uploadId is obtained from initiateMultipartUpload.       obsClient.listParts({            Bucket : 'bucketname',            Key: 'objectname',            UploadId : 'upload id from initiateMultipartUpload',            PartNumberMarker : partNumberMarker       }, function (err, result) {            if(err){                console.log('Error-->' + err);            }else{                console.log('Status-->' + result.CommonMsg.Status);                if(result.CommonMsg.Status < 300 && result.InterfaceResult){                    for(var i in result.InterfaceResult.Parts){                        console.log('Part['+ i +']:');                        // Part number, specified upon uploading                        console.log('PartNumber-->' + result.InterfaceResult.Parts[i]['PartNumber']);                        // Time when the part was last uploaded                        console.log('LastModified-->' + result.InterfaceResult.Parts[i]['LastModified']);                        // Part ETag                        console.log('ETag-->' + result.InterfaceResult.Parts[i]['ETag']);                        // Part size                        console.log('Size-->' + result.InterfaceResult.Parts[i]['Size']);                    }                    if(result.InterfaceResult.IsTruncated === 'true'){                        listAll(result.InterfaceResult.NextPartNumberMarker);                    }                }            }       });};listAll();
```

Listing Multipart Uploads

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

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.)
MaxUploads	Maximum number of multipart uploads listed in the response body. The value ranges from **1** to **1000**. If the value exceeds **1000**, only 1,000 multipart uploads are returned.
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

```
// Create an instance of ObsClient.var obsClient = new ObsClient({    // Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK before storing them in the configuration file or environment variables. In this example, the AK and SK are stored in environment variables. Before running the code in this example, configure environment variables AccessKeyID and SecretAccessKey.    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,    secret_access_key: process.env.SecretAccessKey,    server: 'https://your-endpoint'});obsClient.listMultipartUploads({       Bucket : 'bucketname'}, function (err, result) {       if(err){             console.log('Error-->' + err);       }else{             console.log('Status-->' + result.CommonMsg.Status);             if(result.CommonMsg.Status < 300 && result.InterfaceResult) {                   for(var i in result.InterfaceResult.Uploads){                         console.log('Uploads[' + i + ']');                         console.log('UploadId-->' + result.InterfaceResult.Uploads[i]['UploadId']);                         console.log('Key-->' + result.InterfaceResult.Uploads[i]['Key']);                         console.log('Initiated-->' + result.InterfaceResult.Uploads[i]['Initiated']);                   }             }       }});
```

NOTE

- A maximum of 1,000 multipart uploads can be listed each time. If a bucket contains more than 1,000 multipart uploads, **InterfaceResult.IsTruncated** in the response is **true**, indicating not all uploads were listed. In such case, you can use **InterfaceResult.NextKeyMarker** and **InterfaceResult.NextUploadIdMarker** to obtain the start position for the next listing.
- If you want to obtain all multipart uploads in a bucket, you can list them in paging mode.

- Listing all multipart uploads

```
// Create an instance of ObsClient.var obsClient = new ObsClient({    // Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK before storing them in the configuration file or environment variables. In this example, the AK and SK are stored in environment variables. Before running the code in this example, configure environment variables AccessKeyID and SecretAccessKey.    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.    // Obtain an AK/SK pair on the management console.
```

```
access_key_id: process.env.AccessKeyID,    secret_access_key: process.env.SecretAccessKey,    server: 'https://
your-endpoint'});var listAll = function (keyMarker, uploadIdMarker)
{       obsClient.listMultipartUploads({              Bucket : 'bucketname',              KeyMarker :
keyMarker,              UploadIdMarker : uploadIdMarker       }, function (err, result) {              if(err)
{                   console.log('Error-->' + err);              }else{                   console.log('Status-->' +
result.CommonMsg.Status);                   if(result.CommonMsg.Status < 300 && result.InterfaceResult)
{                       for(var i in result.InterfaceResult.Uploads){                                   console.log('Uploads['
+ i + ']');                                 console.log('UploadId-->' + result.InterfaceResult.Uploads[i]
['UploadId']);                                  console.log('Key-->' + result.InterfaceResult.Uploads[i]
['Key']);                                 console.log('Initiated-->' + result.InterfaceResult.Uploads[i]
['Initiated']);                                 }                                   if(result.InterfaceResult.IsTruncated
=== 'true'){                                    listAll(result.InterfaceResult.NextKeyMarker,
result.InterfaceResult.NextUploadIdMarker);                                 }                   }           }       });}listAll();
```

8.8 Configuring Lifecycle Management

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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
the configuration file or environment variables. In this example, the AK/SK are stored in environment
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// When uploading an object, set the object to expire after 30 days.
obsClient.putObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        Body : 'Hello OBS',
        Expires : 30
}, function(err, result){
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});

// When initializing a multipart upload, set the object to expire 60 days after combination.
obsClient.initiateMultipartUpload({
        Bucket : 'bucketname',
        Key : 'objectname',
        ContentType : 'text/plain',
        Expires : 60
}, function(err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                console.log('UploadId-->' + result.InterfaceResult.UploadId);
            }
        }
});
```


NOTE

- Use the **Expires** parameter to specify expiration time for the object.
- The previous mode specifies the time duration in days after which an object will expire. The OBS server automatically clears expired objects.
- The object expiration time set in the preceding method takes precedence over the bucket lifecycle rule.

8.9 Performing an Appendable Upload

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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Create an appendable object. The start position must be 0.
obsClient.appendObject({
        Bucket:'bucketname',
        Key:'objectname',
        Position : 0,
        Body : 'Hello OBS'
}).then(function(result){
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('NextPosition-->' + result.InterfaceResult.NextPosition);
        }
        // Append data to the appendable object.
        obsClient.appendObject({
            Bucket:'bucketname',
            Key:'objectname',
            Position : result.InterfaceResult.NextPosition,
            Body : 'Hello OBS Again'
        }, function(err, result2){
            if(err){
                console.error('Error-->' + err);
            }else{
                console.log('Status-->' + result2.CommonMsg.Status);
                if(result2.CommonMsg.Status < 300 && result2.InterfaceResult){
                    console.log('NextPosition-->' + result2.InterfaceResult.NextPosition);
                }
            }
        });

       // Use the API for obtaining object properties to get the start position for next appending.
        obsClient.getObjectMetadata({
            Bucket:'bucketname',
            Key:'objectname',
        }).then(function(result3){
            console.log('Status-->' + result3.CommonMsg.Status);
            if(result3.CommonMsg.Status < 300 && result3.InterfaceResult){
                console.log('RequestId-->' + result3.InterfaceResult.RequestId);
                console.log('NextPosition-->' + result3.InterfaceResult.NextPosition);
```



```
            }
        }).catch(function(err){
            console.error('err:' + err);
        });

}).catch(function(err){
    console.error('err:' + err);
});
```

NOTE

- Use the **Position** parameter to specify the start position for next appending and set it to **0** when you create an appendable object.
- Objects uploaded using **ObsClient.putObject**, referred to as normal objects, can overwrite objects uploaded using **ObsClient.appendObject**, referred to as appendable objects. Data cannot be appended to an appendable object anymore once the object has been overwritten by a normal object.
- When you upload an object for the first time in append mode, an exception will be reported (HTTP status code **409**) if an object of the same name exists.
- The ETag returned for an appendable upload is the ETag for the uploaded content, rather than that of the whole object.
- Data appended each time can be up to 5 GB, and 10000 times of appendable uploads can be performed on a single object.
- After an appendable upload is complete successfully, you can use **InterfaceResult.NextPosition** obtained from the returned result or call **ObsClient.getObjectMetadata**, to get the location for next appending.

8.10 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

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

obsClient.CopyPart({
        Bucket : destBucketName,
        Key : destObjectKey,
        // Set the part number, which ranges from 1 to 10000.
        PartNumber : 1,
```



```
    // Set the upload ID.
    UploadId : 'upload id from initiateMultipartUpload',
    // Set the source object to be copied.
    CopySource : sourceBucketName + '/' + sourceObjectKey,
    // Set the to-be-copied range of the source object.
    CopySourceRange : 'bytes=0-100'
}, function(err, result) {
    if (err) {
        console.log('Error-->' + err);
    } else {
        console.log('Status-->' + result.CommonMsg.Status);
        if (result.CommonMsg.Status < 300 && result.InterfaceResult) {
            console.log('RequestId-->' + result.InterfaceResult.RequestId);
            console.log('LastModified-->' + result.InterfaceResult.LastModified);
            console.log('ETag-->' + result.InterfaceResult.ETag);
        }
    }
});
```

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.

8.11 Performing a Resumable Upload

The principle of resumable upload is to divide the files to be uploaded into several parts and upload them separately. The upload results are recorded in the resumable upload result in real time. The upload success result is returned only when all parts are successfully uploaded. If not all parts are successfully uploaded, an error code is returned in the callback function to remind the user to call the API again for upload.

CAUTION

- The total size of files uploaded by the resumable upload API must be larger than 100 KB.
- Refreshing browser pages will lead to invalid uploads. In such case, you need to upload the file again.

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

Parameter	Description
Bucket	Bucket name
Key	Object name
RequestDate	Request time **NOTE** When the parameter type is **String**, the value must comply with the ISO8601 or RFC822 standards.

Parameter	Description
SourceFile	Source file to be uploaded (The browser must support **FileReader**.)
UploadCheckpoint	Resumable upload recording object, which can be obtained through **ResumeCallback**.
PartSize	Part size, in bytes. The value range is **100 KB** to **5 GB**. The default value is **9 MB**.
TaskNum	Maximum number of parts that can be concurrently uploaded. The default value is **1**.
ProgressCallback	Callback function for obtaining the upload progress **NOTE** This callback function contains the following parameters in sequence: Number of uploaded bytes, total bytes, and used time (unit: second).
EventCallback	Callback function for obtaining the upload event **NOTE** The callback function contains the following parameters: Event type, event parameter, and event result.
ResumeCallback	Callback function for obtaining the resumable upload control parameters. **NOTE** • The callback function contains the following parameters: the control parameter for canceling the resumable upload task and the resumable upload record object. • You can invoke the cancel method of the control parameter for canceling the resumable upload task to suspend a resumable upload task.

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

var cp;
var hook;
obsClient.uploadFile({
        Bucket : 'bucketname',
        Key : 'objectname',
        SourceFile : document.getElementById('input-file').files[0],
        PartSize : 9 * 1024 * 1024,
```

```
    ProgressCallback : function(transferredAmount, totalAmount, totalSeconds){
        console.log(transferredAmount * 1.0 / totalSeconds / 1024);
        console.log(transferredAmount * 100.0 / totalAmount);
        if(hook && (transferredAmount / totalAmount) > 0.5){
            // Suspend a resumable download task.
           // hook.cancel();
        }
    },
    EventCallback : function(eventType, eventParam, eventResult){
        // Handle event response.
    },
    ResumeCallback : function(resumeHook, uploadCheckpoint){
        // Obtain the control parameters for canceling resumable upload.
        hook = resumeHook;
        // Record a breakpoint.
        cp = uploadCheckpoint;
    }
}, function(err, result){
   console.error('Error-->' + err);
   // If an error occurs, call the resumable upload API again to continue the upload task.
   if(err){
      obsClient.uploadFile({
        UploadCheckpoint : cp,
        ProgressCallback : function(transferredAmount, totalAmount, totalSeconds){
            console.log(transferredAmount * 1.0 / totalSeconds / 1024);
            console.log(transferredAmount * 100.0 / totalAmount);
        },
        EventCallback : function(eventType, eventParam, eventResult){
         // Handle event response.
        },
       }, function(err, result){
        if(err){
            console.error('Error-->' + err);
        }else{
            if(result.CommonMsg.Status < 300){
              console.log('RequestId-->' + result.InterfaceResult.RequestId);
              console.log('Bucket-->' + result.InterfaceResult.Bucket);
              console.log('Key-->' + result.InterfaceResult.Key);
              console.log('Location-->' + result.InterfaceResult.Location);
            }else{
              console.log('Code-->' + result.CommonMsg.Code);
              console.log('Message-->' + result.CommonMsg.Message);
            }
        }
      });
   }else {
        console.log('Status-->' + result.CommonMsg.Status);
        if (result.CommonMsg.Status < 300 && result.InterfaceResult) {
            console.log('RequestId-->' + result.InterfaceResult.RequestId);
        }
   }
});
```

NOTE

- The API for resumable upload, which is implemented based on **multipart upload**, is an encapsulated and enhanced version of multipart upload.
- The breakpoint record object can be obtained through **ResumeCallback**. The sourceFile field in the resumable upload record object indicates the file to be uploaded. This field needs to be set again after the browser is restarted.

8.12 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.createPostSignatureSync** to generate request parameters for a browser-based upload.

Step 1 Call **ObsClient.createPostSignatureSync** 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.

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Set parameters for the form.
var formParams = {
            // Set the object ACL to public read.
            'x-obs-acl': obsClient.enums.AclPublicRead,
            // Set the MIME type for the object.
            'content-type': 'text/plain'
};

// Set the validity period for the browser-based upload request, in seconds.
var expires = 3600;

var res = obsClient.createPostSignatureSync({Expires:expires, FormParams: formParams});

// Obtain the request parameters.
console.log('\t' + res.Policy);
console.log('\t' + res.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**.

9 Object Download

9.1 Object Download Overview

OBS BrowserJS SDK provides abundant APIs for object download in the following methods:

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

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

9.2 Performing a Text-Based Download

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

The example code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getObject({
        Bucket : 'bucketname',
        Key : 'objectname'
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
```

```
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // Read the object content.
                console.log('Object Content:');
                console.log(result.InterfaceResult.Content);
            }
        }
});
```

NOTE

- In the returned result of a text-based download, **InterfaceResult.Content** is a String object.

9.3 Performing a Binary Download

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        SaveByType : 'arraybuffer'
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // Obtain the object length.
                console.log('Object Length:\n');
                console.log(result.InterfaceResult.Content.byteLength);
            }
        }
});
```

NOTE

- Set the **SaveByType** parameter to **arraybuffer** to use binary download.
- In the returned result of a binary download, **InterfaceResult.Content** is an instance of **ArrayBuffer**.

9.4 Performing a File-Based Download

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

The example code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
```



```
the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        SaveByType : 'file'
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // Obtain the download path of the file.
                console.log('Download Path:');
                console.log(result.InterfaceResult.Content.SignedUrl);
            }
        }
});
```

NOTE

- Set the **SaveByType** parameter to **file** to generate the path for saving the to-be-downloaded file.

9.5 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        // Specify the download range.
        Range : 'bytes=0-1000'
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // Obtain the object content.
```



```
            console.log('Object Content:');
            console.log(result.InterfaceResult.Content);
        }
    }
});
```

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.

9.6 Obtaining Download Progresses

You can set the callback function to obtain download progress.

This example downloads **objectname** from **bucketname** and uses **ProgressCallback** to monitor the download progress.

The example code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

var callback = function(transferredAmount, totalAmount, totalSeconds){
    // Obtain the average download rate (KB/s).
    console.log(transferredAmount * 1.0 / totalSeconds / 1024);
    // Obtain the download progress in percentage.
    console.log(transferredAmount * 100.0 / totalAmount);
};

obsClient.getObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        ProgressCallback: callback
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // Obtain the object content.
                console.log('Object Content:');
                console.log(result.InterfaceResult.Content);
            }
        }
});
```

NOTE

- Only text-based and binary download support obtaining the download progress.

9.7 Performing a Conditioned Download

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

You can set the following conditions:

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

NOTE

- The ETag of 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 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 with error code **304 Not Modified** returned.

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment
```



```
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        IfModifiedSince : 'Thu, 31 Dec 2015 16:00:00 GMT'
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // Obtain the object content.
                console.log('Object Content:');
                console.log(result.InterfaceResult.Content);
            }
        }
});
```

9.8 Rewriting Response Headers

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

Parameter	Description
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. **NOTICE** If you set **SaveByType** to **file**, rewriting **Content-Disposition** is not allowed. You need to use **a temporary URL** to download the file and rewrite **Content-Disposition**.
ResponseContentEncoding	Rewrites **Content-Encoding** in HTTP/HTTPS responses.

Rewriting the ContentType Response Header

The following code shows how to rewrite the **ContentType** response header:


```
// Create an instance of ObsClient.var obsClient = new ObsClient({    // Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK before storing them in the configuration file or environment variables. In this example, the AK and SK are stored in environment variables. Before running the code in this example, configure environment variables AccessKeyID and SecretAccessKey.    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,    secret_access_key: process.env.SecretAccessKey,    server: 'https://your-endpoint'});obsClient.getObject({       Bucket : 'bucketname',       Key : 'objectname',       ResponseContentType : 'image/jpeg'}, function (err, result) {       if(err){            console.error('Error-->' + err);       }else{            console.log('Status-->' + result.CommonMsg.Status);            if(result.CommonMsg.Status < 300 && result.InterfaceResult){                // Obtain the rewritten response headers.                console.log(result.InterfaceResult.ContentType);            }       }});
```

Rewriting the ContentDisposition Response Header

The following code shows how to rewrite the **ContentDisposition** response header when **SaveByType** is set to **file**:

```
// Create an instance of ObsClient.var obsClient = new ObsClient({    // Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK before storing them in the configuration file or environment variables. In this example, the AK and SK are stored in environment variables. Before running the code in this example, configure environment variables AccessKeyID and SecretAccessKey.    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,    secret_access_key: process.env.SecretAccessKey,    server: 'https://your-endpoint'});var res = obsClient.createSignedUrlSync({    Method: 'GET',    Bucket: 'bucketName',    Key: 'objectKey',    Expires: 3600,    QueryParams: {'response-content-disposition' : 'attachment; filename=name'}});console.log(res.SignedUrl)
```

9.9 Obtaining Customized Metadata

The customized metadata of an object will be returned after the object is successfully downloaded.

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

The example code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Download the object and obtain the customized metadata.
obsClient.getObject({
        Bucket : 'bucketname',
        Key : 'objectname',
}, function (err, result) {
        if(err){
                console.error('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                    console.log('Metadata-->' + JSON.stringify(result.InterfaceResult.Metadata['property']));
                }
        }
});
```


NOTE

- To obtain the customized metadata of an object, you need to add the additional headers that are allowed to be carried in responses to the CORS configurations. For example, you can add **x-obs-meta-property** to enable the query of **property**.

9.10 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 in OBS BrowserJS SDK
Expedited restore	Data can be restored within 1 to 5 minutes.	ObsClient.enums.RestoreTierExpedited
Standard restore	Data can be restored within 3 to 5 hours. This is the default option.	ObsClient.enums.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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Restore an object in the Cold storage class.
obsClient.restoreObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        Days : 1,
        Tier : obsClient.enums.RestoreTierExpedited
}, function (err, result) {
        if(err){
                console.error('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);

// Wait for the object to be restored.
```



```
        setTimeout(function () {

            // Download the object and obtain the object content.
            obsClient.getObject({
                Bucket : 'bucketname',
                Key : 'objectname'
            }, function (err, result) {
                if(err){
                    console.error('Error-->' + err);
                }else{
                    console.log('Status-->' + result.CommonMsg.Status);
                    if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                        // Obtain the object content.
                        console.log('Object Content:');
                        console.log(result.InterfaceResult.Content);
                    }
                }
            });
        }, 6 * 60 * 1000);
    }
});
```

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.

10 Object Management

10.1 Configuring Object Metadata

You can call **ObsClient.setObjectMetadata** to set object attributes, including system-defined metadata (**CacheControl**, **ContentDisposition**, **ContentType**, **StorageClass**, and **Expires**).

This example configures the **Content-Disposition** metadata for **objectname** in **bucketname**.

The example code is as follows:

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process object, so you need to use a module bundler like webpack to define environment variables.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});
obsClient.setObjectMetadata({
        Bucket : 'bucketname',
        Key : 'objectname',
        Content-Disposition : 'attachment;filename=1.txt',
        MetadataDirective : 'REPLACE_NEW'
}, function (err, result){
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

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

10.2 Obtaining Object Properties

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

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

The example code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getObjectMetadata({
        Bucket : 'bucketname',
        Key : 'objectname'
}, function (err, result){
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                console.log(result.InterfaceResult.ContentType);
                console.log(result.InterfaceResult.ContentLength);
                console.log(result.InterfaceResult.Metadata['property']);
            }
        }
});
```

NOTE

- To obtain the customized metadata of an object, you need to add the additional headers that are allowed to be carried in responses to the CORS configurations. For example, you can add **x-obs-meta-property** to enable the query of **property**.

10.3 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 a pre-defined ACL.
3. Call ObsClient.setObjectAcl to specify a user-defined ACL.

Specifying a Pre-defined ACL During Object Upload

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.putObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        Body : 'Hello OBS',
        // Set the object ACL to public read.
        ACL : obsClient.enums.AclPublicRead
}, function (err, result){
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

Setting a Pre-defined ACL for an Object

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.setObjectAcl({
        Bucket : 'bucketname',
        Key : 'objectname',
        // Set the object ACL to private read and write.
        ACL : obsClient.enums.AclPrivate
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```


Setting a User-defined Object ACL

The following code shows how to set a user-defined ACL for an object:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

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.enums.PermissionFullControl},
                // Grant the READ permission to all users.
             { Grantee: {Type : 'Group', URI : obsClient.enums.GroupAllUsers}, Permission : obsClient.enums.PermissionRead}
        ]
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

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.enums.GroupAllUsers

Obtaining an Object ACL

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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});
```



```
obsClient.getObjectAcl({
      Bucket : 'bucketname',
      Key : 'objectname'
}, function (err, result) {
      if(err){
        console.error('Error-->' + err);
      }else{
        console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                  console.log('Owner[ID]-->' + result.InterfaceResult.Owner.ID);
                  for(var i in result.InterfaceResult.Grants){
                        console.log('Grant[' + i + ']:');
                        console.log('Grantee[ID]-->' + result.InterfaceResult.Grants[i]['Grantee']['ID']);
                        console.log('Grantee[URI]-->' + result.InterfaceResult.Grants[i]['Grantee']['URI']);
                        console.log('Permission-->' + result.InterfaceResult.Grants[i]['Permission']);
                  }
            }
      }
});
```

10.4 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 to be listed in the response body. The value ranges from **1** to **1000**. If the specified value exceeds **1000**, only 1,000 objects are returned.
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 1,000 objects can be returned.

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.listObjects({
        Bucket : 'bucketname'
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                    for(var j in result.InterfaceResult.Contents){
                            console.log('Contents[' + j +  ']:');
                            console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                            console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']['ID']);
                     }
                }
        }
});
```

NOTE

- A maximum of 1,000 objects can be listed each time. If a bucket contains more than 1,000 objects, **InterfaceResult.IsTruncated** in the response is **true**, indicating not all objects were listed. In such case, you can use **InterfaceResult.NextMarker** to obtain the start position for the 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.listObjects({
        Bucket : 'bucketname',
        // Set the number of objects to be listed to 100.
        MaxKeys : 100
```



```
}, function (err, result) {
    if(err){
      console.error('Error-->' + err);
    }else{
      console.log('Status-->' + result.CommonMsg.Status);
          if(result.CommonMsg.Status < 300 && result.InterfaceResult){
              for(var j in result.InterfaceResult.Contents){
                    console.log('Contents[' + j +  ']:');
                    console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']['ID']);
              }
          }
    }
});
```

Listing Objects by Specifying a Prefix

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.listObjects({
      Bucket : 'bucketname',
      // Set the prefix to prefix and the number of objects to be listed to 100.
      MaxKeys : 100,
      Prefix : 'prefix'
}, function (err, result) {
      if(err){
        console.error('Error-->' + err);
      }else{
        console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                for(var j in result.InterfaceResult.Contents){
                      console.log('Contents[' + j +  ']:');
                      console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                      console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']
['ID']);
                 }
            }
      }
});
```

Listing Objects by Specifying the Start Position

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
```

```
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.listObjects({
        Bucket : 'bucketname',
        // Specify that 100 objects whose names follow test in lexicographical order will be listed.
        MaxKeys : 100,
        Marker : 'test'
}, function (err, result) {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                    for(var j in result.InterfaceResult.Contents){
                            console.log('Contents[' + j +  ']:');
                            console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                            console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']['ID']);
                     }
                }
        }
});
```

Listing All Objects in Paging Mode

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

var listAll = function (marker) {
        obsClient.listObjects({
            Bucket : 'bucketname',
            // Set the number of parts displayed per page to 100.
            MaxKeys : 100,
            Marker : marker
        }, function (err, result) {
            if(err){
                console.error('Error-->' + err);
            }else{
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                    for(var j in result.InterfaceResult.Contents){
                            console.log('Contents[' + j +  ']:');
                            console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                            console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']
['ID']);
                    }
                    if(result.InterfaceResult.IsTruncated === 'true'){
                        listAll(result.InterfaceResult.NextMarker);
                    }
                }
            }
        });
};

listAll();
```

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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

var listAll = function (marker) {
      obsClient.listObjects({
            Bucket : 'bucketname',
            MaxKeys : 1000,
            // Set the prefix of the folders to dir/.
            Prefix : 'dir/'
      }, function (err, result) {
            if(err){
                  console.error('Error-->' + err);
            }else{
                  console.log('Status-->' + result.CommonMsg.Status);
                  if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                        for(var j in result.InterfaceResult.Contents){
                              console.log('Contents[' + j +  ']:');
                              console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                              console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']['ID']);
                        }
                        if(result.InterfaceResult.IsTruncated === 'true'){
                              listAll(result.InterfaceResult.NextMarker);
                        }
                  }
            }
      });
};

listAll();
```

Listing All Objects According to Folders in a Bucket

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});
```



```
obsClient.listObjects({
    Bucket: 'bucketname',
    // Set folder isolators to slashes (/).
    Delimiter: '/'
}, function (err, result) {
    if(!err && result.CommonMsg.Status < 300){
        console.log('Objects in the root directory:');
        for(var j in result.InterfaceResult.Contents){
            console.log('\tKey-->' + result.InterfaceResult.Contents[j]['Key']);
            console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']
['ID']);
        }
        var listObjectsByPrefix = function (commonPrefixes){
            for(var i in commonPrefixes){
                var prefix = commonPrefixes[i]['Prefix'];
                obsClient.listObjects({
                    Bucket: 'bucketname',
                    Delimiter: '/',
                    Prefix: prefix
                }, function (err, result) {
                    if(!err && result.CommonMsg.Status < 300){
                        console.log('Objects in folder:');
                        for(var j in result.InterfaceResult.Contents){
                            console.log('\tKey-->' + result.InterfaceResult.Contents[j]['Key']);
                            console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']
['ID']);
                        }
                        console.log('\n');
                        if(result.InterfaceResult.CommonPrefixes &&
result.InterfaceResult.CommonPrefixes.length > 0){
                            listObjectsByPrefix(result.InterfaceResult.CommonPrefixes);
                        }
                    }
                });
            }
        };
        listObjectsByPrefix(result.InterfaceResult.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, **InterfaceResult.Contents** includes the objects in the folder and **InterfaceResult.CommonPrefixes** includes the sub-folders in the folder.

10.5 Deleting Objects

Deleting a Single Object

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

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
the configuration file or environment variables. In this example, the AK/SK are stored in environment
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
```

```
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.deleteObject({
        Bucket: 'bucketname',
        Key : 'objectname'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

Deleting Objects in a Batch

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

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

- In verbose mode (default mode), the returned response includes the deletion result of each requested object.
- In quiet mode, the returned response includes only results of objects failed to be deleted.

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.deleteObjects({
        Bucket: 'bucketname',
        // Set the response mode to verbose.
        Quiet : false,
        Objects : [{Key:'objectname1'},{Key:'objectname2'}, {Key : 'objectname3'}]
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
              // Obtain the successfully deleted objects.
              console.log('Deleteds:');
              for(var i in result.InterfaceResult.Deleteds){
                    console.log('Deleted[' + i + ']:');
                    console.log('Key-->'+result.InterfaceResult.Deleteds[i]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.Deleteds[i]['VersionId']);
              }
              // Obtain information about objects that were not deleted.
```



```
            console.log('Errors:');
            for(var i in result.InterfaceResult.Errors){
                console.log('Error[' + i + ']:');
                console.log('Key-->' + result.InterfaceResult.Errors[i]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Errors[i]['VersionId']);
            }
        }
    }
});
```

NOTE

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

10.6 Copying an Object

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

You can call **ObsClient.copyObject** to copy an object. When copying an object, you can specify properties and ACL for it.

NOTE

- If the source object to be copied is in the Cold storage class, you must restore it first.

Copying an Object Directly

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.copyObject({
        Bucket: 'destbucketname',
        Key : 'destobjectname',
        CopySource:'sourcebucketname/sourceobjectname'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

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

Rewriting Object Properties

The following sample code shows how to rewrite object properties.

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.copyObject({
    Bucket: 'destbucketname',
    Key : 'destobjectname',
    CopySource:'sourcebucketname/sourceobjectname',
    ContentType : 'image/jpeg',
    StorageClass : obsClient.enums.StorageClassWarm,
    Metadata:{'property':'property-value'},
    MetadataDirective : ObsClient.enums.ReplaceMetadata
}, function (err, result) {
      if(err){
            console.log('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
      }
});
```

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.enums.ReplaceMetadata** (rewrite) or **ObsClient.enums.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, the copy will fail with an error code returned.

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.

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

NOTE

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

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.copyObject({
    Bucket: 'destbucketname',
    Key : 'destobjectname',
    CopySource:'sourcebucketname/sourceobjectname',
    CopySourceIfModifiedSince : 'Thu, 31 Dec 2015 16:00:00 GMT',
    CopySourceIfNoneMatch : 'none-match-etag'
}, function (err, result) {
      if(err){
            console.log('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
      }
});
```

Rewriting an Object ACL

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
```

```
the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.copyObject({
    Bucket: 'destbucketname',
    Key : 'destobjectname',
    CopySource:'sourcebucketname/sourceobjectname',
    // Rewrite the object ACL to public read during the copy.
    ACL : obsClient.enums.AclPublicRead
}, function (err, result) {
      if(err){
            console.log('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
      }
});
```

NOTE

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

11 Temporarily Authorized Access

11.1 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
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

Operation	HTTP Method	Special Operator (Sub-resource)	Bucket Name Required	Object Name Required
GET Bucket location	GET	location	Yes	No
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

Operation	HTTP Method	Special Operator (Sub-resource)	Bucket Name Required	Object Name Required
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
GET Bucket cors	GET	cors	Yes	No
DELETE Bucket cors	DELETE	cors	Yes	No
PUT Object	PUT	N/A	Yes	Yes
Append Object	POST	append	Yes	Yes

Operation	HTTP Method	Special Operator (Sub-resource)	Bucket Name Required	Object Name Required
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
	POST	restore	Yes	Yes

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

Step 1 Call **ObsClient.createSignedUrlSync** 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 object upload, download, listing, and deletion.

Uploading an Object

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});
//Make a PUT request to upload an object.
var bucketName = 'bucketname';
var objectKey = 'objectname';
var method = 'PUT';
var headers = {
        'Content-Type' : 'text/plain'
}
var res = obsClient.createSignedUrlSync({
        Method : method,
        Bucket : bucketName,
        Key : objectKey,
        Expires : 3600,
        Headers : headers
});
var content = 'Hello OBS';

var reopt = {
        method : method,
        url : res.SignedUrl,
        withCredentials: false,
        headers : res.ActualSignedRequestHeaders || {},
        validateStatus: function(status){
                return status >= 200;
        },
        maxRedirects : 0,
        responseType : 'text',
        data : content,
};

axios.request(reopt).then(function (response) {
        if(response.status < 300){
                console.log('Creating object using temporary signature succeed.');
        }else{
                console.log('Creating object using temporary signature failed!');
                console.log('status:' + response.status);
                console.log('\n');
        }
        console.log(response.data);
        console.log('\n');
}).catch(function (err) {
        console.log('Creating object using temporary signature failed!');
        console.log(err);
        console.log('\n');
});
```


Downloading an Object

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

//Make a GET request to download an object.
var bucketName = 'bucketname';
var objectKey = 'objectname';
var method = 'GET';

var res = obsClient.createSignedUrlSync({
        Method : method,
        Bucket : bucketName,
        Key : objectKey,
        Expires : 3600,
});

var reopt = {
        method : method,
        url : res.SignedUrl,
        withCredentials: false,
        headers : res.ActualSignedRequestHeaders || {},
        validateStatus: function(status){
            return status >= 200;
        },
        maxRedirects : 0,
        responseType : 'text',
};

axios.request(reopt).then(function (response) {
        if(response.status < 300){
            console.log('Getting object using temporary signature succeed.');
        }else{
            console.log('Getting object using temporary signature failed!');
            console.log('status:' + response.status);
            console.log('\n');
        }
        console.log(response.data);
        console.log('\n');
}).catch(function (err) {
        console.log('Getting object using temporary signature failed!');
        console.log(err);
        console.log('\n');
});
```

Listing Objects

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
```



```
    server: 'https://your-endpoint'
});

// Make a GET request to obtain the object list.
var bucketName = 'bucketname';
var method = 'GET';

var res = obsClient.createSignedUrlSync({
        Method : method,
        Bucket : bucketName,
        Expires : 3600,
});

var reopt = {
        method : method,
        url : res.SignedUrl,
        withCredentials: false,
        headers : res.ActualSignedRequestHeaders || {},
        validateStatus: function(status){
                return status >= 200;
        },
        maxRedirects : 0,
        responseType : 'text',
};

axios.request(reopt).then(function (response) {
        if(response.status < 300){
                console.log('Listing object using temporary signature succeed.');
        }else{
                console.log('Listing object using temporary signature failed!');
                console.log('status:' + response.status);
                console.log('\n');
        }
        console.log(response.data);
        console.log('\n');
}).catch(function (err) {
        console.log('Listing object using temporary signature failed!');
        console.log(err);
        console.log('\n');
});
```

Deleting an Object

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Make a DELETE request to delete the object.
var bucketName = 'bucketname';
var objectKey = 'objectname';
var method = 'DELETE';

var res = obsClient.createSignedUrlSync({
        Method : method,
        Bucket : bucketName,
        Key : objectKey,
        Expires : 3600,
});
```



```
var reopt = {
    method : method,
    url : res.SignedUrl,
    withCredentials: false,
    headers : res.ActualSignedRequestHeaders || {},
    validateStatus: function(status){
        return status >= 200;
    },
    maxRedirects : 0,
    responseType : 'text',
};

axios.request(reopt).then(function (response) {
    if(response.status < 300){
        console.log('Deleting object using temporary signature succeed.');
    }else{
        console.log('Deleting object using temporary signature failed!');
        console.log('status:' + response.status);
        console.log('\n');
    }
    console.log(response.data);
    console.log('\n');
}).catch(function (err) {
    console.log('Deleting object using temporary signature failed!');
    console.log(err);
    console.log('\n');
});
```

Initiating a Multipart Upload

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Make a POST request to initiate the multipart upload.
var bucketName = 'bucketname';
var objectKey = 'objectname';
var method = 'POST';

var res = obsClient.createSignedUrlSync({
    Method : method,
    Bucket : bucketName,
    Key : objectKey,
    Expires : 3600,
    SpecialParam: "uploads"
});

var reopt = {
    method : method,
    url : res.SignedUrl,
    withCredentials: false,
    headers : res.ActualSignedRequestHeaders || {},
    validateStatus: function(status){
        return status >= 200;
    },
    maxRedirects : 0,
    responseType : 'text',
};
```



```
axios.request(reopt).then(function (response) {
    if(response.status < 300){
        console.log('Initiate multipart upload using temporary signature succeed.');
    }else{
        console.log('Initiate multipart upload using temporary signature failed!');
        console.log('status:' + response.status);
        console.log('\n');
    }
    console.log(response.data);
    console.log('\n');
}).catch(function (err) {
    console.log('Initiate multipart upload using temporary signature failed!');
    console.log(err);
    console.log('\n');
});
```

Uploading a Part

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Make a PUT request to upload a part.
var bucketName = 'bucketname';
var objectKey = 'objectname';
var method = 'PUT';
var headers = {
    'Content-Type' : 'text/plain'
}

var res = obsClient.createSignedUrlSync({
    Method : method,
    Bucket : bucketName,
    Key : objectKey,
    Expires : 3600,
    Headers: headers
    QueryParams: {
        // Specify the part number.
        'partNumber': '1',
        // Specify the ID of the multipart upload, which is returned in the response for initiating the multipart upload.
        'uploadId': '000001648453845DBB78F2340DD4*****',
    }
});

var content = 'Hello OBS';

var reopt = {
    method : method,
    url : res.SignedUrl,
    withCredentials: false,
    headers : res.ActualSignedRequestHeaders || {},
    validateStatus: function(status){
        return status >= 200;
    },
    maxRedirects : 0,
    responseType : 'text',
    data : content,
};
```



```
axios.request(reopt).then(function (response) {
    if(response.status < 300){
        console.log('Upload part using temporary signature succeed.');
    }else{
        console.log('Upload part using temporary signature failed!');
        console.log('status:' + response.status);
        console.log('\n');
    }
    console.log(response.data);
    console.log('\n');
}).catch(function (err) {
    console.log('Upload part upload using temporary signature failed!');
    console.log(err);
    console.log('\n');
});
```

Assembling Parts

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Make a POST request to assemble the uploaded parts.
var bucketName = 'bucketname';
var objectKey = 'objectname';
var method = 'POST';
var headers = {
    'Content-Type' : 'application/xml'
}

var res = obsClient.createSignedUrlSync({
    Method : method,
    Bucket : bucketName,
    Key : objectKey,
    Expires : 3600,
    Headers: headers,
    QueryParams: {
        // Specify the ID of the multipart upload, which is returned in the response for initiating the multipart upload.
        'uploadId': '000001648453845DBB78F2340DD4*****',
    }
});

var content = "<CompleteMultipartUpload>";
content += "<Part>";
content += "<PartNumber>1</PartNumber>";
content += "<ETag>da6a0d097e307ac52ed9b4ad551801fc</ETag>";
content += "</Part>";
content += "<Part>";
content += "<PartNumber>2</PartNumber>";
content += "<ETag>da6a0d097e307ac52ed9b4ad551801fc</ETag>";
content += "</Part>";
content += "</CompleteMultipartUpload>";

var reopt = {
    method : method,
    url : res.SignedUrl,
    withCredentials: false,
```



```
    headers : res.ActualSignedRequestHeaders || {},
    validateStatus: function(status){
        return status >= 200;
    },
    maxRedirects : 0,
    responseType : 'text',
    data : content,
};

axios.request(reopt).then(function (response) {
    if(response.status < 300){
        console.log('Complete multipart upload using temporary signature succeed.');
    }else{
        console.log('Complete multipart upload using temporary signature failed!');
        console.log('status:' + response.status);
        console.log('\n');
    }
    console.log(response.data);
    console.log('\n');
}).catch(function (err) {
    console.log('Complete multipart upload using temporary signature failed!');
    console.log(err);
    console.log('\n');
});
```

NOTE

1. 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 operators, and the **QueryParams** parameter to specify the query parameters.
2. **ActualSignedRequestHeaders** in the response to a backend URL request may carry **Host**, which is not required for a browser-based upload. You need to delete **Host** from **ActualSignedRequestHeaders**. Otherwise, an error is reported.

12 Versioning Management

12.1 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:

- **Setting Versioning Status for a Bucket**
- **Viewing Versioning Status of a Bucket**
- **Deleting Versioning Objects**

12.2 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
```



```
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Enable versioning.
obsClient.setBucketVersioning({
        Bucket : 'bucketname',
        VersionStatus : 'Enabled'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});

// Suspend versioning.
obsClient.setBucketVersioning({
        Bucket : 'bucketname',
        VersionStatus : 'Suspended'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

Use the **VersionStatus** parameter to specify the versioning status.

12.3 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Enable versioning for a bucket.
obsClient.getBucketVersioning({
        Bucket : 'bucketname'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                console.log('VersionStatus-->' + result.InterfaceResult.VersionStatus);
            }
```



```
    }
});
```

NOTE

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

12.4 Obtaining a Versioning Object

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

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Set the version ID to obtain a versioning object.
obsClient.getObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        VersionId : 'versionid'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                console.log('Content-->' + result.InterfaceResult.Content);
            }
        }
});
```

NOTE

versionId indicates an object version ID, which can be obtained by **listing object versions by specifying a prefix**.

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

12.5 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 **sourcebucket** to **destbucketname** as **destobjectname**.

The example code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables
```

```
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.copyObject({
        Bucket : 'destbucketname',
        Key : 'destobjectname',
        // Set the version ID of the object to be copied.
        CopySource : 'sourcebucket/sourceobjectname?versionId=versionid'
}, function (err, result) {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

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

12.6 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:

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
the configuration file or environment variables. In this example, the AK/SK are stored in environment
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.restoreObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        VersionId : 'versionid',
        Days : 1,
        // Restore a versioned object at an expedited speed.
        Tier : obsClient.enums.RestoreTierExpedited
}, function (err, result) {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

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.

12.7 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 listed versioning objects. The value ranges from **1** to **1000**. If the specified value exceeds **1000**, only 1,000 versioning objects are returned.
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 1,000 versioning objects can be returned.

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
```

```
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.listVersions({
        Bucket : 'bucketname'
}, function (err, result) {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                        // Obtain versioning objects.
                        for(var j in result.InterfaceResult.Versions){
                                console.log('Version[' + j +  ']:');
                                console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                                console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                                console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
                        }
                        // Obtain delete markers.
                        for(var i in result.InterfaceResult.DeleteMarkers){
                                console.log('DeleteMarker[' + i +  ']:');
                                console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                                console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                                console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
                        }
                }
        }
});
```

NOTE

- A maximum of 1,000 object versions can be listed each time. If a bucket contains more than 1,000 object versions, **InterfaceResult.IsTruncated** in the response is **true**, indicating not all object versions were listed. In such case, you can use **InterfaceResult.NextKeyMarker** and **InterfaceResult.NextVersionIdMarker** to obtain the start position for the 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.listVersions({
```

```
    Bucket : 'bucketname',
    MaxKeys : 100
}, function (err, result) {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // Obtain versioning objects.
            for(var j in result.InterfaceResult.Versions){
                console.log('Version[' + j +  ']:');
                console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
            }
            // Obtain delete markers.
            for(var i in result.InterfaceResult.DeleteMarkers){
                console.log('DeleteMarker[' + i +  ']:');
                console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
            }
        }
    }
});
```

Listing Versioning Objects by Specifying a Prefix

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Set the prefix to prefix and the number to 100.
obsClient.listVersions({
    Bucket : 'bucketname',
    MaxKeys : 100,
    Prefix : 'prefix'
}, function (err, result) {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // Obtain versioning objects.
            for(var j in result.InterfaceResult.Versions){
                console.log('Version[' + j +  ']:');
                console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
            }
            // Obtain delete markers.
            for(var i in result.InterfaceResult.DeleteMarkers){
                console.log('DeleteMarker[' + i +  ']:');
                console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
             }
```



```
        }
    }
});
```

Listing Versioning Objects by Specifying the Start Position

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// List 100 versioning objects whose names are following test in the lexicographical order.
obsClient.listVersions({
        Bucket : 'bucketname',
        MaxKeys : 100,
        KeyMarker : 'test'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // Obtain versioning objects.
                for(var j in result.InterfaceResult.Versions){
                    console.log('Version[' + j +  ']:');
                    console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
                }
                // Obtain delete markers.
                for(var i in result.InterfaceResult.DeleteMarkers){
                    console.log('DeleteMarker[' + i +  ']:');
                    console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
                }
            }
        }
});
```

Listing All Versioning Objects in Paging Mode

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});
```



```
var listAll = function (keyMarker, versionIdMarker) {
    obsClient.listVersions({
        Bucket : 'bucketname',
        MaxKeys : 100,
        KeyMarker : keyMarker,
        VersionIdMarker : versionIdMarker
    }, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // Obtain versioning objects.
                for(var j in result.InterfaceResult.Versions){
                    console.log('Version[' + j +  ']:');
                    console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
                }
                // Obtain delete markers.
                for(var i in result.InterfaceResult.DeleteMarkers){
                    console.log('DeleteMarker[' + i +  ']:');
                    console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
                }
                if(result.InterfaceResult.IsTruncated === 'true'){
                    listAll(result.InterfaceResult.NextKeyMarker,
result.InterfaceResult.NextVersionIdMarker);
                }
            }
        }
    });
};

listAll();
```

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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
the configuration file or environment variables. In this example, the AK/SK are stored in environment
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

var listAll = function (keyMarker, versionIdMarker) {
    obsClient.listVersions({
        Bucket : 'bucketname',
        MaxKeys : 100,
        KeyMarker : keyMarker,
        VersionIdMarker : versionIdMarker,
        // Set the prefix of the folders to dir/.
        Prefix : 'dir/'
    }, function (err, result) {
```

```
            if(err){
                console.log('Error-->' + err);
            }else{
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                    // Obtain versioning objects.
                    for(var j in result.InterfaceResult.Versions){
                        console.log('Version[' + j +  ']:');
                        console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                        console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                        console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
                    }
                    // Obtain delete markers.
                    for(var i in result.InterfaceResult.DeleteMarkers){
                        console.log('DeleteMarker[' + i +  ']:');
                        console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                        console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                        console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
                    }
                    if(result.InterfaceResult.IsTruncated === 'true'){
                        listAll(result.InterfaceResult.NextKeyMarker,
result.InterfaceResult.NextVersionIdMarker);
                    }
                }
            }
        });
};

listAll();
```

Listing All Versioning Objects According to Folders in a Bucket

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
the configuration file or environment variables. In this example, the AK/SK are stored in environment
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.listVersions({
        Bucket : 'bucketname',
        // Set folder isolators to slashes (/).
        Delimiter : '/'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Objects in the root directory:');
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // Obtain versioning objects.
                for(var j in result.InterfaceResult.Versions){
                    console.log('Version[' + j +  ']:');
                    console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
                }
                // Obtain delete markers.
                for(var i in result.InterfaceResult.DeleteMarkers){
                    console.log('DeleteMarker[' + i +  ']:');
                    console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
```

```
                console.log('VersionId-->'+ result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
            }
        }

        var listVersionsByPrefix = function (commonPrefixes) {
            for(var n in commonPrefixes){
                obsClient.listVersions({
                    Bucket : 'bucketname',
                    Delimiter : '/',
                    Prefix: commonPrefixes[n]['Prefix']
                }, function (err, result) {
                    console.log('Objects in folder:');
                    if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                      // Obtain versioning objects.
                      for(var j in result.InterfaceResult.Versions){
                          console.log('Version[' + j +  ']:');
                          console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                          console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                          console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
                      }
                      // Obtain delete markers.
                      for(var i in result.InterfaceResult.DeleteMarkers){
                          console.log('DeleteMarker[' + i +  ']:');
                          console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                          console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]
['VersionId']);
                          console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']
['ID']);
                      }
                      console.log('\n');
                      if(result.InterfaceResult.CommonPrefixes &&
result.InterfaceResult.CommonPrefixes.length > 0){
                          listVersionsByPrefix(result.InterfaceResult.CommonPrefixes);
                      }
                    }
                });
            }
        };
        listVersionsByPrefix(result.InterfaceResult.CommonPrefixes);
    }
});
```

NOTE

- The previous sample code does not include scenarios where the number of versioning 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, **InterfaceResult.Versions** includes the versioning objects in the folder, **InterfaceResult.DeleteMarkers** includes the delete markers in the folder, and **InterfaceResult.CommonPrefixes** includes the sub-folders in the folder.

12.8 Setting or Obtaining an Object Version ACL

Setting an ACL for an Object Version

You can call **ObsClient.setObjectAcl** and specify the **VersionId** parameter to configure an ACL for an object version. Sample code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
```

```
the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.setObjectAcl({
        Bucket : 'bucketname',
        Key : 'objectname',
        VersionId : 'versionid',
        // Set the object version ACL to public read by specifying a pre-defined ACL.
        ACL : obsClient.enums.AclPublicRead
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});

obsClient.setObjectAcl({
        Bucket : 'bucketname',
        Key : 'objectname',
        VersionId : 'versionid',
        // Set the object owner.
        Owner:{'ID':'ownerid'},
        Grants:[
                // Grant the READ and WRITE_ACP permission to all users.
                { Grantee:{Type : 'Group', URI : obsClient.enums.GroupAllUsers}, Permission : obsClient.enums.PermissionRead},
                { Grantee:{Type : 'Group', URI : obsClient.enums.GroupAllUsers}, Permission : obsClient.enums.PermissionWriteAcp}
        ]
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

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.enums.GroupAllUsers

Obtaining the ACL of an Object Version

You can call **ObsClient.getObjectAcl** and specify the **VersionId** parameter to obtain the ACL of an object version. Sample code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
```

```
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getObjectAcl({
        Bucket : 'bucketname',
        Key : 'objectname',
        VersionId : 'versionid'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                console.log('Owner[ID]-->' + result.InterfaceResult.Owner.ID);
                console.log('Owner[Name]-->' + result.InterfaceResult.Owner.Name);
                for(var i in result.InterfaceResult.Grants){
                    console.log('Grant[' + i + ']:');
                    console.log('Grantee[ID]-->' + result.InterfaceResult.Grants[i]['Grantee']['ID']);
                    console.log('Grantee[URI]-->' + result.InterfaceResult.Grants[i]['Grantee']['URI']);
                    console.log('Permission-->'+ result.InterfaceResult.Grants[i]['Permission']);
                }
            }
        }
});
```

12.9 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
'
});

obsClient.deleteObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        VersionId : 'versionid'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
```

```
    }
});
```

Deleting Versioning Objects in a Batch

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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.deleteObjects({
        Bucket: 'bucketname',
        // Set the response mode to verbose.
        Quiet : false,
        Objects : [{Key:'objectname1', VersionId : 'version1'},{Key:'objectname2', VersionId : 'version2'}, {Key : 'objectname3', VersionId : 'version3'}]
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // Obtain the successfully deleted objects.
                console.log('Deleteds:');
                for(var i in result.InterfaceResult.Deleteds){
                    console.log('Deleted[' + i + ']:');
                    console.log('Key-->'+result.InterfaceResult.Deleteds[i]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.Deleteds[i]['VersionId']);
                }
                // Obtain information about objects that were not deleted.
                console.log('Errors:');
                for(var j in result.InterfaceResult.Errors){
                    console.log('Error[' + j + ']:');
                    console.log('Key-->' + result.InterfaceResult.Errors[j]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.Errors[j]['VersionId']);
                }
            }
        }
});
```


13 Lifecycle Management

13.1 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.

13.2 Setting Lifecycle Rules

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

Setting an Object Transition Policy

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.setBucketLifecycle({
        Bucket: 'bucketname',
        Rules:[
            {
                ID:'rule1',Prefix:'prefix1',Status:'Enabled',
                // Specify that objects whose names contain the specified prefix will be transitioned to OBS Warm 30 days after creation.
                Transitions:[{StorageClass: obsClient.enums.StorageClassWarm, Days:30}],
                // Specify that objects whose names contain the specified prefix will be transitioned to OBS Cold after being noncurrent for 30 days.
                NoncurrentVersionTransitions:[{StorageClass: obsClient.enums.StorageClassCold, NoncurrentDays : 30}]
            },
            {
                ID:'rule2',Prefix:'prefix2',Status:'Enabled',
                // Specify the date when objects whose names contain the specified prefix will be transitioned to OBS Warm.
                Transitions:[{StorageClass: obsClient.enums.StorageClassWarm, Date: '2018-10-31T00:00:00Z'}],
            }
        ]
}, function(err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

Setting an Object Expiration Time

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.setBucketLifecycle({
        Bucket: 'bucketname',
        Rules:[
            {
                ID:'rule1',Prefix:'prefix1',Status:'Enabled',
                 // Specify that objects whose names contain the specified prefix will expire 60 days after creation.
                Expiration:{Days:60},
                 // Specify that objects whose names contain the specified prefix will expire after changing into noncurrent versions for 60 days.
                NoncurrentVersionExpiration:{NoncurrentDays : 60}
            },
            {
                ID:'rule2',Prefix:'prefix2',Status:'Enabled',
                // Specify when the objects whose names contain the specified 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'},
            }
        ]
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

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

13.3 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
'
});
```

```
obsClient.getBucketLifecycle({
      Bucket: 'bucketname'
}, function (err, result) {
      if(err){
            console.log('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                  for(var i=0;i<result.InterfaceResult.Rules.length;i++){
                        console.log('Rule[' + i + ']:');
                        console.log('ID-->' + result.InterfaceResult.Rules[i]['ID']);
                        console.log('Prefix-->' + result.InterfaceResult.Rules[i]['Prefix']);
                        console.log('Status-->' + result.InterfaceResult.Rules[i]['Status']);
                        for(var j=0;j<result.InterfaceResult.Rules[i]['Transitions'].length;j++){
                              console.log('Transition[' + j + ']:');
                              console.log('Transition[StorageClass]-->' + result.InterfaceResult.Rules[i]['Transitions']
[j]['StorageClass']);
                              console.log('Transition[Date]-->' + result.InterfaceResult.Rules[i]['Transitions'][j]
['Date']);
                              console.log('Transition[Days]-->' + result.InterfaceResult.Rules[i]['Transitions'][j]
['Days']);                        }
                        console.log('Expiration[Date]-->' + result.InterfaceResult.Rules[i]['Expiration']['Date']);
                        console.log('Expiration[Days]-->' + result.InterfaceResult.Rules[i]['Expiration']['Days']);
                        for(var k=0;k<result.InterfaceResult.Rules[i]['NoncurrentVersionTransitions'].length;k++){
                              console.log('NoncurrentVersionTransition[' + k + ']:');
                              console.log('NoncurrentVersionTransition[StorageClass]-->' +
result.InterfaceResult.Rules[i]['NoncurrentVersionTransitions'][k]['StorageClass']);
                              console.log('NoncurrentVersionTransition[NoncurrentDays]-->' +
result.InterfaceResult.Rules[i]['NoncurrentVersionTransitions'][k]['NoncurrentDays']);                        }
                        console.log('NoncurrentVersionExpiration[NoncurrentDays]-->' +
result.InterfaceResult.Rules[i]['NoncurrentVersionExpiration']['NoncurrentDays']);
                  }
            }
      }
});
```

NOTE

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

13.4 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.deleteBucketLifecycle({
      Bucket: 'bucketname'
}, function (err, result) {
```

```
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

NOTE

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

14 Access Logging

14.1 Logging Overview

OBS allows you to configure access logging for buckets. After the configuration, access to buckets will be recorded in 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.

14.2 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:

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
```



```
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Configure logging for the bucket.
obsClient.setBucketLogging({
    Bucket:'bucketname',
    // Name of the agency created on IAM.
    Agency: 'Agency name'
    LoggingEnabled:{
        // Name of the bucket for storing the generated log file.
        TargetBucket: 'logBucketName',
        // Specify the name prefix of the generated log file.
        TargetPrefix: 'logs/',
    }
}, function(err, result) {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

NOTE

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

Setting ACLs for Objects to Be Logged

Sample code:

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in
the configuration file or environment variables. In this example, the AK/SK are stored in environment
variables for identity authentication. Before running this example, configure environment variables
AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module
bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Configure logging for the bucket.
obsClient.setBucketLogging({
    Bucket:'bucketname',
    // Name of the OBS agency created by the owner of the target bucket on IAM.
    Agency: 'Agency name',
    LoggingEnabled:{
        // Name of the bucket for storing the generated log file.
        TargetBucket: 'LogBucketName',
        // Specify the name prefix of the generated log file.
        TargetPrefix: 'logs/',
        TargetGrants:[
            // Grant all users the READ permission on the logs.
            { Grantee:
{Type:'Group',URI:obsClient.enums.GroupAllUsers},Permission:obsClient.enums.PermissionRead },
            // Grant all users the WRITE permission on the logs.
            { Grantee:
{Type:'Group',URI:obsClient.enums.GroupAllUsers},Permission:obsClient.enums.PermissionWrite }
        ]
     }
}, function(err, result) {
```



```
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

14.3 Viewing Bucket Logging

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:

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getBucketLoggingConfiguration({
        Bucket:'bucketname'
}, function (err, result) {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                        if(result.InterfaceResult.LoggingEnabled){
                                console.log('TargetBucket-->' + result.InterfaceResult.LoggingEnabled.TargetBucket);
                                console.log('TargetPrefix-->' + result.InterfaceResult.LoggingEnabled.TargetPrefix);
                        }
                        for(var i in result.InterfaceResult.LoggingEnabled.TargetGrants){
                                console.log('Grant[' + i + ']:');
                                console.log('Grantee[ID]-->' + result.InterfaceResult.LoggingEnabled.TargetGrants[i]['Grantee']['ID']);
                                console.log('Grantee[URI]-->' + result.InterfaceResult.LoggingEnabled.TargetGrants[i]['Grantee']['URI']);
                                console.log('Permission-->' + result.InterfaceResult.LoggingEnabled.TargetGrants[i]['Permission']);
                        }
                }
        }
});
```

NOTE

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

14.4 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:

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.setBucketLoggingConfiguration({
        Bucket:'bucketname',
        LoggingEnabled : {}
}, function (err, result) {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

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

15 Static Website Hosting

15.1 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.

15.2 Website File Hosting

You can perform the following to implement website file hosting:

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

Step 2 Set the object ACL to public read.

Step 3 Access the object using a browser.

----End

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
```



```
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Upload an object.
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.enums.AclPublicRead
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

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

15.3 Setting Website Hosting

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

Configuring the Default Homepage and Error Pages

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.setBucketWebsite({
        Bucket: 'bucketname',
        // Configure the default homepage.
        IndexDocument:{Suffix:'index.html'},
        // Configure the error pages.
        ErrorDocument:{Key:'error.html'}
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```


Configuring Redirection Rules

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

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', KeyPrefixEquals: 'keyprefix'},
                    Redirect:{Protocol:'http',ReplaceKeyWith:'replacekeyprefix', HostName: 'www.example.com', HttpRedirectCode: '305'}
                }
        ]
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

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

Configuring Redirection for All Requests

Sample code:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.setBucketWebsite({
        Bucket: 'bucketname',
        RedirectAllRequestsTo : {HostName : 'www.example.com', Protocol : 'http'}
}, function (err, result) {
        if(err){
```

```
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

NOTE

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

15.4 Viewing Website Hosting Settings

You can call **ObsClient.getBucketWebsite** to view the hosting configuration of a bucket.

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

The example code is as follows:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.getBucketWebsite({
        Bucket: 'bucketname'
}, function (err, result) {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                console.log('RedirectAllRequestsTo:');
                console.log('HostName-->' + result.InterfaceResult.RedirectAllRequestsTo['HostName']);
                console.log('Protocol-->' + result.InterfaceResult.RedirectAllRequestsTo['Protocol']);
                console.log('IndexDocument[Suffix]-->' + result.InterfaceResult.IndexDocument['Suffix']);
                console.log('ErrorDocument[Key]-->' + result.InterfaceResult.ErrorDocument['Key']);
                console.log('RoutingRules:');
                for(var i in result.InterfaceResult.RoutingRules){
                    console.log('RoutingRule[' + i + ']:');
                    var RoutingRule = result.InterfaceResult.RoutingRules[i];
                    console.log('Condition[HttpErrorCodeReturnedEquals]-->' + RoutingRule['Condition']['HttpErrorCodeReturnedEquals']);
                     console.log('Condition[KeyPrefixEquals]-->' + RoutingRule['Condition']['KeyPrefixEquals']);
                    console.log('Redirect[HostName]-->' + RoutingRule['Redirect']['HostName']);
                    console.log('Redirect[HttpRedirectCode]-->' + RoutingRule['Redirect']['HttpRedirectCode']);
                    console.log('Redirect[Protocol]-->' + RoutingRule['Redirect']['Protocol']);
                    console.log('Redirect[ReplaceKeyPrefixWith]-->' + RoutingRule['Redirect']['ReplaceKeyPrefixWith']);
                    console.log('Redirect[ReplaceKeyWith]-->' + RoutingRule['Redirect']['ReplaceKeyWith']);
                }
            }
        }
});
```

NOTE

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

15.5 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:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.deleteBucketWebsite({
        Bucket: 'bucketname'
}, function (err, result) {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

NOTE

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

16 Troubleshooting

16.1 HTTP Status Codes

The OBS server complies with the HTTP standard. After an API is called, the OBS server returns a standard HTTP status code. The following tables list the categories of HTTP status codes and the common HTTP status codes in OBS.

- Categories of HTTP status codes

Category	Description
1XX	Informational response. A request is received by the server and the server requires the requester to continue the operation. This category is usually invisible to the client.
2XX	Success. The operation is received and processed successfully.
3XX	Redirection. Further operations to complete the request are required.
4XX	Client errors. The request contains a syntax error, or the request cannot be implemented.
5XX	Server errors. An error occurs when the server is processing the request.

- Common HTTP status codes in OBS and their meanings

HTTP Status Code	Description	Possible Cause
400 Bad Request	Incorrect request parameter.	• Invalid request parameter. • The consistency check fails after the client request carries MD5. • An invalid parameter is transferred when the SDK is used. • An invalid bucket name is used.
403 Forbidden	Access denied.	• The signature of the request does not match. Generally, the error is caused by incorrect AK/SK. • The account does not have the permission to access the requested resource. • The account is in arrears. • The bucket space is insufficient when a quota is set for the bucket. • Invalid AK • The time difference between the client and the server is too large. That is, the time of the server where the client is located is not synchronized with the time of the NTP service.
404 Not Found	The requested resource does not exist.	• The bucket does not exist. • The object does not exist. • The bucket policy configuration does not exist. For example, the bucket CORS configuration or bucket policy configuration does not exist. • The multipart upload does not exist.
405 Method Not Allowed	The request method is not supported.	The requested method or feature is not supported in the region where the bucket resides.

HTTP Status Code	Description	Possible Cause
408 Request Timeout	Request timed out.	The Socket connection between the server and client timed out.
409 Conflict	Request conflicts occur.	• Buckets of the same name are created in different regions. • Deletion of a non-empty bucket is attempted.
500 Internal Server Error	Internal server error.	An internal error occurs on the server side.
503 Service Unavailable	Service unavailable.	The server cannot be accessed temporarily.

16.2 OBS Server-Side Error Codes

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

HTTP Status Code	Error Code	Error Message	Solution
301 Moved Permanently	PermanentRedirect	The requested bucket can be accessed only through the specified address. Send subsequent requests to the address.	Send the request to the returned redirection address.
301 Moved Permanently	WebsiteRedirect	The website request lacks **bucketName**.	Put the bucket name in the request and try again.
307 Moved Temporarily	TemporaryRedirect	Temporary redirection. If the DNS is updated, the request is redirected to the bucket.	The system automatically redirects the request or sends the request to the redirection address.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	BadDigest	The specified value of **Content-MD5** does not match the value received by OBS.	Check whether the MD5 value carried in the header is the same as that calculated by the message body.
400 Bad Request	BadDomainName	Invalid domain name.	Use a valid domain name.
400 Bad Request	BadRequest	Invalid request parameter.	Modify the parameter according to the error details returned in the message body.
400 Bad Request	CustomDomainAreadyExist	The configured domain already exists.	It has been configured and does not need to be configured again.
400 Bad Request	CustomDomainNotExist	Delete the domain that does not exist.	It is not configured or has been deleted. You do not need to delete it.
400 Bad Request	EntityTooLarge	The size of the object uploaded using the POST method exceeds the upper limit.	Modify the conditions specified in the policy when posting the object or reduce the object size.
400 Bad Request	EntityTooSmall	The size of the object uploaded using the POST method does not reach the lower limit.	Modify the conditions specified in the policy when posting the object or increase the object size.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	IllegalLocation-ConstraintException	A request without **Location** is sent for creating a bucket in a non-default region.	Send the bucket creation request to the default region, or send the request with the **Location** of the non-default region.
400 Bad Request	IncompleteBody	No complete request body is received due to network or other problems.	Upload the object again.
400 Bad Request	IncorrectNumber-OfFilesInPost Request	Each POST request must contain one file to be uploaded.	Carry a file to be uploaded.
400 Bad Request	InvalidArgument	Invalid parameter.	Modify the parameter according to the error details returned in the message body.
400 Bad Request	InvalidBucket	The bucket to be accessed does not exist.	Try another bucket name.
400 Bad Request	InvalidBucketNam e	The bucket name specified in the request is invalid, which may have exceeded the maximum length, or contain special characters that are not allowed.	Try another bucket name.
400 Bad Request	InvalidLocation-Constraint	The specified **Location** in the bucket creation request is invalid or does not exist.	Correct the **Location** in the bucket creation request.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	InvalidPart	One or more specified parts are not found. The parts may not be uploaded or the specified entity tags (ETags) do not match the parts' ETags.	Merge the parts correctly according to the ETags.
400 Bad Request	InvalidPartOrder	Parts are not listed in ascending order by part number.	Sort the parts in ascending order and merge them again.
400 Bad Request	InvalidPolicyDocument	The content of the form does not meet the conditions specified in the policy document.	Modify the policy in the constructed form according to the error details in the message body and try again.
400 Bad Request	InvalidRedirectLocation	Invalid redirect location.	Specify the correct IP address.
400 Bad Request	InvalidRequest	Invalid request.	Modify the parameter according to the error details returned in the message body.
400 Bad Request	InvalidRequestBody	The request body is invalid. The request requires a message body but no message body is uploaded.	Upload the message body in the correct format.
400 Bad Request	InvalidTargetBucketForLogging	The delivery group has no ACL permission for the target bucket.	Configure the target bucket ACL and try again.
400 Bad Request	KeyTooLongError	The provided key is too long.	Use a shorter key.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	MalformedACLError	The provided XML file is in an incorrect format or does not meet format requirements.	Use the correct XML format to retry.
400 Bad Request	MalformedError	The XML format in the request is incorrect.	Use the correct XML format to retry.
400 Bad Request	MalformedLoggingStatus	The XML format of **Logging** is incorrect.	Use the correct XML format to retry.
400 Bad Request	MalformedPolicy	The bucket policy failed the check.	Modify the bucket policy according to the error details returned in the message body.
400 Bad Request	MalformedQuotaError	The Quota XML format is incorrect.	Use the correct XML format to retry.
400 Bad Request	MalformedXML	An XML file of a configuration item is in incorrect format.	Use the correct XML format to retry.
400 Bad Request	MaxMessageLengthExceeded	Copying an object does not require a message body in the request.	Remove the message body and retry.
400 Bad Request	MetadataTooLarge	The size of the metadata header has exceeded the upper limit.	Reduce the size of the metadata header.
400 Bad Request	MissingRegion	No region contained in the request and no default region defined in the system.	Carry the region information in the request.
400 Bad Request	MissingRequestBodyError	An empty XML file is sent as a request.	Provide the correct XML file.

HTTP Status Code	Error Code	Error Message	Solution
400 Bad Request	MissingRequired-Header	A required header is missing in the request.	Provide the required header.
400 Bad Request	MissingSecurity-Header	A required header is missing in the request.	Provide the required header.
400 Bad Request	TooManyBuckets	You have attempted to create more buckets than allowed.	Delete some buckets and try again.
400 Bad Request	TooManyCustomDomains	Too many user accounts are configured.	Delete some user accounts and try again.
400 Bad Request	TooManyWrongSignature	The request is rejected due to high-frequency errors.	Replace AK and try again.
400 Bad Request	UnexpectedContent	The request requires a message body which is not carried by the client, or the request does not require a message body but the client carries the message body.	Try again according to the instruction.
400 Bad Request	UserKeyMustBeSpecified	This operation is only available to special users.	Contact the technical support.
403 Forbidden	AccessDenied	Access denied, because the request does not carry a date header or the header format is incorrect.	Provide a correct date header in the request.

HTTP Status Code	Error Code	Error Message	Solution
403 Forbidden	AccessForbidden	Insufficient permission. No CORS rule is configured for the bucket or the CORS rule does not match.	Modify the CORS configuration of the bucket or send the matched OPTIONS request based on the CORS configuration of the bucket.
403 Forbidden	AllAccessDisabled	You have no permission to perform the operation. The bucket name is forbidden.	Try another bucket name.
403 Forbidden	DeregisterUserId	The user has been deregistered.	Top up or re-register.
403 Forbidden	InArrearOrInsufficientBalance	The subscriber owes fees or the account balance is insufficient, and the subscriber does not have the permission to perform an operation.	Top up the account.
403 Forbidden	InsufficientStorageSpace	Insufficient storage space.	If the quota is exceeded, increase quota or delete some objects.
403 Forbidden	InvalidAccessKeyId	The access key ID provided by the customer does not exist in the system.	Provide correct access key ID.
403 Forbidden	NotSignedUp	You have not registered with the system.	Register OBS.
403 Forbidden	RequestTimeTooSkewed	The request time and the server's time differ a lot.	Check whether the difference between the client time and the current time is too large.

HTTP Status Code	Error Code	Error Message	Solution
403 Forbidden	SignatureDoesNotMatch	The provided signature in the request does not match the signature calculated by OBS.	Check your secret access key and signature calculation method.
403 Forbidden	Unauthorized	You have not been authenticated in real name.	Authenticate your real name and try again.
404 Not Found	NoSuchBucket	The specified bucket does not exist.	Create a bucket and perform the operation again.
404 Not Found	NoSuchBucketPolicy	No bucket policy exists.	Configure a bucket policy.
404 Not Found	NoSuchCORSConfiguration	No CORS configuration exists.	Configure CORS first.
404 Not Found	NoSuchCustomDomain	The requested user domain does not exist.	Set a user domain first.
404 Not Found	NoSuchKey	The specified key does not exist.	Upload the object first.
404 Not Found	NoSuchLifecycle-Configuration	The requested lifecycle rule does not exist.	Configure a lifecycle rule first.
404 Not Found	NoSuchUpload	The specified multipart upload does not exist. The upload ID does not exist or the multipart upload job has been aborted or completed.	Use the existing part or reinitialize the part.
404 Not Found	NoSuchVersion	The specified version ID does not match any existing version.	Use a correct version ID.

HTTP Status Code	Error Code	Error Message	Solution
404 Not Found	NoSuchWebsiteConfiguration	The requested website does not exist.	Configure the website first.
405 Method Not Allowed	MethodNotAllowed	The specified method is not allowed against the requested resource. The message "Specified method is not supported." is returned.	The method is not allowed.
408 Request Timeout	RequestTimeout	The socket connection to the server has no read or write operations within the timeout period.	Check the network and try again, or contact technical support.
409 Conflict	BucketAlreadyExists	The requested bucket name already exists. The bucket namespace is shared by all users of OBS. Select another name and retry.	Try another bucket name.
409 Conflict	BucketAlreadyOwnedByYou	Your previous request for creating the named bucket succeeded and you already own it.	You do not need to create the bucket again.
409 Conflict	BucketNotEmpty	The bucket that you tried to delete is not empty.	Delete the objects in the bucket and then delete the bucket.
409 Conflict	OperationAborted	A conflicting operation is being performed on this resource. Retry later.	Try again later.

HTTP Status Code	Error Code	Error Message	Solution
409 Conflict	ServiceNotSupported	The request method is not supported by the server.	Not supported by the server. Contact technical support.
411 Length Required	MissingContentLength	The HTTP header Content-Length is not provided.	Provide the Content-Length header.
412 Precondition Failed	PreconditionFailed	At least one of the specified preconditions is not met.	Modify according to the condition prompt in the returned message body.
416 Client Requested Range Not Satisfiable	InvalidRange	The requested range cannot be obtained.	Retry with the correct range.
500 Internal Server Error	InternalError	An internal error occurs. Retry later.	Contact the technical support.
501 Not Implemented	ServiceNotImplemented	The request method is not implemented by the server.	Not supported currently. Contact the technical support.
503 Service Unavailable	ServiceUnavailable	The server is overloaded or has internal errors.	Try again later or contact the technical support.
503 Service Unavailable	SlowDown	Too frequent requests. Reduce your request frequency.	Reduce your request frequency.

16.3 SDK Common Result Objects

After you call an API in an instance of the **ObsClient** class, a common result object will be returned if the exception information parameter is null. The following table lists the object content:

Field	Type	Description
CommonMsg	Object	Common information generated after the API calling is complete, including HTTP status code and error code.

Field		Type	Description
-	Status	Number	HTTP status code. If the value is smaller than **300**, the operation succeeds. Otherwise, the operation fails.
	Code	String	Error code on the OBS server. If **Status** is smaller than **300**, the value is null.
	Message	String	Error description on the OBS server. If **Status** is smaller than **300**, the value is null.
	HostId	String	Requested server ID. If **Status** is smaller than **300**, the value is null.
	RequestId	String	Request ID returned by the OBS server
	Id2	String	Request ID2 returned by the OBS server
	Indicator	String	Detailed error code returned by the OBS server. If **Status** is smaller than **300**, the value is null.
InterfaceResult		Object	Result data generated after the operation is successful. If **Status** is greater than 300, the value is null.
-	RequestId	String	Request ID returned by the OBS server
	Id2	String	Request ID2 returned by the OBS server
	Other fields	For details, see the *OBS BrowserJS SDK API Reference*.	

Sample code for processing public result objects:

```
// Create an instance of ObsClient.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

// Call APIs to perform related operations, for example, downloading an object.
obsClient.getObject({
        Bucket : 'bucketname',
        Key : 'objectname',
}, function (err, result) {
        if(!err){
            if(result.CommonMsg.Status < 300){
                // Obtain the request ID.
                console.log('RequestId-->' + result.InterfaceResult.RequestId);
                // Obtain other parameters.
                console.log('Content-->' + result.InterfaceResult.Content);
            }else{
```



```
                // Obtain Code and Message.
                console.log('Code-->' + result.CommonMsg.Code);
                console.log('Message-->' + result.CommonMsg.Message);
            }
        }
});
```

16.4 Log Analysis

Log Configuration

OBS BrowserJS SDK provides the logging function. You can call **ObsClient.initLog** to enable and configure logging. Sample code is as follows:

```
// Create an ObsClient instance.
var obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // The front-end code does not have the process environment variable, so you need to use a module bundler like webpack to define the process variable.
    // Obtain an AK/SK pair on the management console.    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});

obsClient.initLog({
      level:'warn', // Set the log level.
});
```

NOTE

- Logs printed by the SDK will be displayed in the Console of the developer tool provided by the browser.
- The logging function is disabled by default. You need to enable it if needed.

Log Format

The SDK log format is: Log time|log level|log content. The following are example logs:

```
2018/2/11 PM 9:22:45|info|ListObjects|enter ListObjects...
2018/2/11 PM 9:22:45|info|ListObjects|prepare request parameters ok,then Send request to service start
2018/2/11 PM 9:22:45|info|ListObjects|http cost 19 ms
2018/2/11 PM 9:22:45|info|ListObjects|get response start, statusCode:200
```

Log Level

When current logs cannot be used to troubleshoot system faults, you can change the log level to obtain more information. You can obtain the most information in **debug** logs and the least information in **error** logs.

Log level description:

- **debug**: Debugging level. If this level is set, all logs will be printed.
- **info**: Information level. If this level is set, information about logs of the warn level and time consumed for each HTTP/HTTPS request will be printed.
- **warn**: Warning level. If this level is set, information about logs at the error level and information about partial critical events will be printed.

- **error**: Error level. If this level is set, only error information will be printed.

17 FAQs

17.1 How Can I Set an Object to Be Accessible to Anonymous Users?

To enable anonymous users to access an object, perform the following steps:

Step 1 Set the object ACL to public read by referring to **Managing Object ACLs**.

Step 2 Obtain the URL of the object by referring to **How Do I Obtain an Object URL?** and provide it to anonymous users.

Step 3 An anonymous user can access the object by entering the URL on a browser.

----End

17.2 How Do I Obtain the Static Website Access Address of a Bucket?

After a bucket is configured to work in static website hosting mode, you can use the following method to combine the static website access address of the bucket.

https://*bucket name.static website hosting domain name*

NOTE

17.3 How Do I Obtain an Object URL?

Compose the URL in the format of **https://***Bucket name.Domain name*/*Directory level*/*Object name*.

NOTE

- If the object resides in the root directory of the bucket, its URL does not contain directory levels.

17.4 How to Improve the Speed of Uploading Large Files over the Public Network?

If the size of a file exceeds 100 MB, you are advised to upload the file using multipart upload over the public network. Multipart upload allows uploading a single object as parts separately. Each part is a part of consecutive object data. You can upload parts in any sequence. A part can be reloaded after an upload failure, without affecting other parts. Uploading multiple parts of an object using multiple threads concurrently can greatly improve the transmission efficiency.

For details about the code example, see **Performing a Multipart Upload**. You can also use the **API for resumable upload** provided by the SDK to upload large files.

17.5 How Do I Interact with OBS Without Exposing My AK and SK?

Using BrowserJS SDK to interact with OBS will expose the AK and SK to the frontend, which has security risks. To avoid this problem, the frontend can use the temporarily signed URL generated by the backend to interact with OBS.

The following takes an upload as an example.

```
// Use the Node.js SDK at the backend.
// Import the OBS library.
const ObsClient = require('esdk-obs-nodejs');
// Perform initialization.
const obsClient = new ObsClient({
    // Hard-coded or plaintext AK/SK are risky. For security purposes, encrypt your AK/SK and store them in the configuration file or environment variables. In this example, the AK/SK are stored in environment variables for identity authentication. Before running this example, configure environment variables AccessKeyID and SecretAccessKey.
    // Obtain an AK/SK pair on the management console.
    access_key_id: process.env.AccessKeyID,
    secret_access_key: process.env.SecretAccessKey,
    server: 'https://your-endpoint'
});
// The frontend transfers the following three parameters to the backend for calculating the signature:
// Bucket name
const bucketName = 'bucketName';
// Object name
const objectKey = 'object';
// HTTP request method
const method = 'PUT'
// Add the Content-Type header and specify the file type. text/plain is an example.
const headers = {
    'Content-Type': 'text/plain'
}
// The backend calculates the signed URL and returns it to the frontend.
const res = obsClient.createSignedUrlSync({ Method : method, Bucket : bucketName, Key: objectKey, Expires: 3600, Headers : headers });

// The frontend sends an HTTP request to OBS.
// The frontend uses the Axios library.
const url = res.SignedUrl;
const file = document.getElementById('input[type=file]')[0].file
if (!file) {
   console.log('your file is undefined')
}

axios.put(url, file, { headers })
```

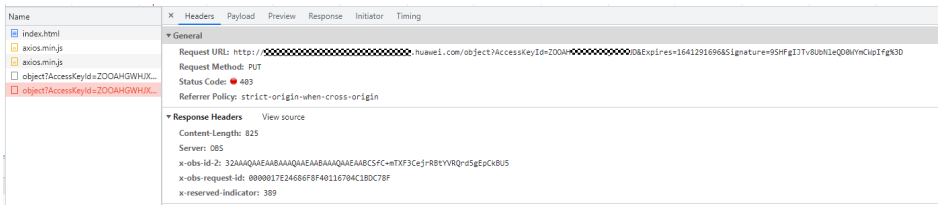


```
.then(res => console.log(res))
.catch(err => console.error(err));
```

CAUTION

When using this solution, you may encounter cross-origin problems. Go through the following list in order, to locate the problem:

1. If no CORS rules are configured, configure one on OBS Console. For details, see **Configuring CORS**.
2. Check whether the signature parameters are correct. In the demo here, for example, the **PUT method of Axios automatically adds a request header**. If this header is not passed in the backend for generating a temporary URL, a cross-origin error occurs, and status code 403 is displayed on the **Network** tab of the browser, as shown below.



17.6 What Do I Do If the Resumable Upload API Reports a "400 InvalidPart" Error?

The error was possibly caused by:

1. a certain part requested not existing.
2. an incorrect ETag of a certain part.

Perform the following steps to locate the reason:

Step 1 Open the developer tool of the browser.

Step 2 Check whether the request body of the API for assembling parts complies with the API specifications.

- If all ETag values in the request body are **undefined**, this indicates the ETag field has not been configured as an extended header of CORS. In this case, the error was caused by reason 2. Refer to **Configuring CORS for a Bucket** to add the ETag field.

Step 3 If the request body complies with the API specifications, proceed to:

- Check whether the list of parts requested contains any parts that do not exist. Assume an object was divided into three parts: **a**, **b**, and **c**, but the list contains an additional part **d**. In this case, the error was caused by reason 1. To solve the problem, delete all parts that do not exist.

- Check whether all ETag values in the request body are the same as those returned by the server. If any values are different, the error was caused by reason 2. To solve the problem, correct all incorrect ETag values.

----End